Livre Blanc

DELPHI 2010 DATASNAP – TOUTES VOS DONNÉES… OÙ VOUS VOULEZ… COMME VOUS VOULEZ…

Bob Swart – Bob Swart Training & Consultancy (eBob42)

Siège social
100 California Street, 12th Floor
San Francisco, California 94111

Siège EMEA
York House
18 York Road
Maidenhead, Berkshire
SL6 1SF, United Kingdom

Siège Asia-Pacific
L7. 313 La Trobe Street
Melbourne VIC 3000
Australia

Copyright © 2009 Bob Swart (aka Dr. Bob - www.drbob42.com). Tous droits réservés.

Ce livre blanc couvre les nouveaux outils et fonctionnalités de l'architecture Delphi 2010 DataSnap.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

1. Historique DataSnap

Précédemment connue sous le nom de MIDAS (Delphi 3), MIDAS II (Delphi 4) et MIDAS III (Delphi 5), cette solution offrait un moyen de développer de puissants modules COM d'accès distant aux données avec des fonctionnalités de connexion TCP/IP, HTTP et (D)COM. Le nom DataSnap est apparu avec Delphi 6 – et son framework est resté relativement intact jusqu'à Delphi 2007.
Depuis Delphi 2009, une version réarchitecturée de DataSnap est proposée supprimant les dépendances avec COM et introduisant un moyen plus léger pour produire des objets et serveurs distants et connectivités client – à l'origine uniquement avec des connexions TCP/IP mais avec la possibilité de construire des clients .NET avec Delphi Prism 2009.
Delphi 2010 repose sur l'architecture DataSnap 2009 et en étend le framework avec de nouvelles fonctionnalités : prise en charge de nouvelles cibles à travers deux nouveaux Assistants (Fiches VCL, Windows Service, Console et également cibles Web telles que ISAPI, CGI ou Débogueur d'application Web), protocoles de transport HTTP(S), authentification HTTP, fonction de rappel client, support REST et JSON, filtres de compression (déjà intégrés) et cryptage.

1.1 Exemple DataSnap – Toutes vos données… *Où vous voulez*…

Je vous engage à largement utiliser les démonstrations et exemples fournis dans ce livre blanc. Delphi prend en charge différents systèmes de bases de données (avec des technologies d'accès telles que DBX4, dbGo for ADO, etc.). Pour simplifier la manipulation des exemples, j'utiliserai DBX4 avec BlackfishSQL comme SGBDR avec la base de données employee.jds située dans le répertoire C:\Documents and Settings\All Users\Documents\RADStudio\7.0\Demos\database\databases\BlackfishSQL sous Windows XP
ou
C:\Users\Public\Documents\ RAD Studio\7.0\Demos\database\databases\BlackfishSQL sous Windows Vista ou Windows 7.
Comme vous le verrez dans les copies d'écran, j'utilise le système d'exploitation Windows 7 Professionnel pour les exemples et Windows Server 2008 Web Edition pour le déploiement des serveurs DataSnap ISAPI.

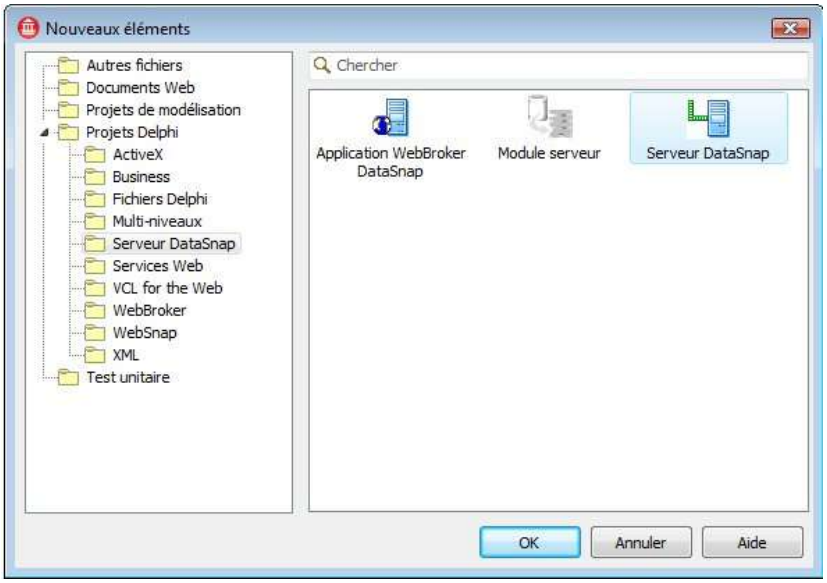
Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

2. Cibles Windows DataSnap… Comme vous voulez…

DataSnap 2010 prend en charge trois différentes cibles Windows : applications Fiches VCL, applications Service et les applications Console. Dans cette section, j'envisagerai les avantages, différences et meilleurs cas d'utilisation de chacune de ces cibles.
Nous construirons un serveur et un client DataSnap et couvrirons les composants TDSServer, TDSServerClass, TDSTCPServerTransport, TDSHTTPService, TDSHTTPWebDispatcher et TDSHTTPServiceAuthenticationManager, les méthodes serveur personnalisées et la classe TDSServerModule.
Nous aborderons différents protocoles de transport (TCP, HTTP) ainsi que leurs effets et avantages respectifs potentiels. Les différentes options de durée de vie des objets serveur DataSnap seront également abordées (serveur, session et invocation) avec leurs effets et des recommandations pratiques. Enfin, nous aborderons certains enjeux de déploiement.

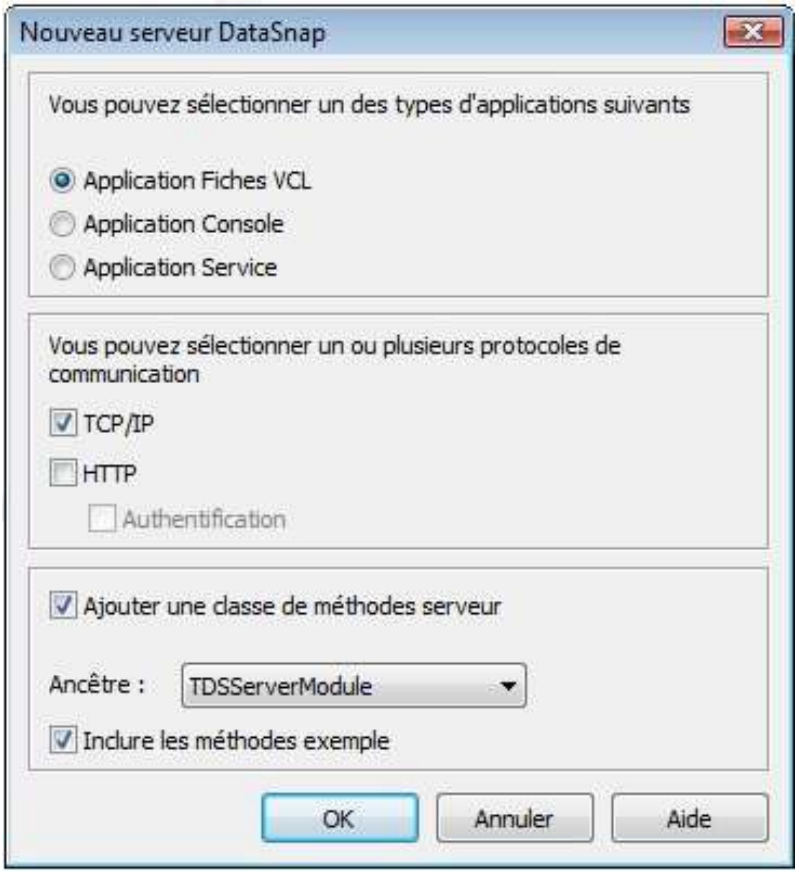
2.1. Exemple de Serveur DataSnap

Il existe deux assistants Serveur DataSnap dans le référentiel d'objets : un pour les projets Windows, l'autre pour les projets WebBroker qui doivent être hébergés sur un serveur Web tel que IIS (Internet Information Services).
Nous débuterons par le premier.
Si vous débutez avec Delphi 2010, vous trouverez les assistants Serveur DataSnap dans le référentiel d'objets (*Fichier* | *Nouveau* | *Autre*). La catégorie serveur DataSnap affiche trois icônes : Serveur DataSnap , Application WebBroker DataSnap et Module serveur.



Effectuez un double-clic sur Serveur DataSnap (nous reviendrons ultérieurement sur les deux autres) pour ouvrir la boîte de dialogue suivante :

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



La première partie de cette boîte de dialogue permet de contrôler la cible du projet. Par défaut il s'agit d'une application graphique VCL (avec un formulaire principal) ; le second choix produit une fenêtre console – idéale pour tracer les requêtes/réponses (avec de simples déclarations writeln pour démontrer ce qui se passe réellement dans l'application serveur). Ces deux types d'applications sont idéales pour des démonstrations et développements initiaux mais finalement moins pratiques pour déployer une application serveur DataSnap. La nouvelle architecture DataSnap n'étant plus basée sur COM, une connexion client entrante ne permet pas de « lancer » une application serveur DataSnap. Par conséquent, pour manipuler les requêtes client entrantes, le serveur DataSnap doit déjà être opérationnel et en fonctionnement. En outre, si vous souhaitez pouvoir traiter les demandes entrantes 24X7, l'application serveur DataSnap doit également être lancée et opérationnelle en permanence. Pour une application Fiches VCL ou Console, cela signifie qu'un compte doit être logué sous Windows pour exécuter l'application serveur DataSnap – ce qui n'est pas idéal. Le troisième choix est préférable dans ce cas : une application de Service Windows qui peut être installée et configurée pour être automatiquement exécutée au démarrage de la machine (sans qu'il soit nécessaire que quelqu'un soit logué sur la machine). L'inconvénient d'une application de service est que, par défaut, elle n'apparaît pas sur le poste client et est également un peu plus complexe à déboguer. Cependant, pour obtenir le meilleur des trois mondes, je démontrerai dans quelques instants comment créer un groupe projet pour une application Serveur DataSnap Fiches VCL, Console et Windows Service partageant les mêmes méthodes serveur personnalisées – permettant ainsi de compiler (et

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

déployer) la même application serveur pour trois cibles différentes en cas de besoin.
La deuxième partie de la boîte de dialogue permet de sélectionner les protocoles de communication. Par comparaison à DataSnap 2009, il est désormais possible de choisir des communications et authentifications HTTP. Pour plus de flexibilité, nous sélectionnerons toutes les options pour pouvoir utiliser TCP/IP et HTTP pour les communications et HTTP pour l'authentification.
La dernière partie est configurée correctement par défaut ; elle permet de générer une classe de méthodes serveur et même de choisir le type : TPersistent, TDataModule ou TDSServerModule. Le dernier choix est le meilleur car il active dès le départ RTTI pour les méthodes (dans certaines circonstances TDataModule ou éventuellement – si vous n'utilisez pas de jeux de données ni de contrôles non-visuels – TPersistent peuvent suffire).
Le snippet suivant de DSServer.pas indique la relation entre TDSServerModule et TProviderDataModule – dérivé à son tour de TDataModule.

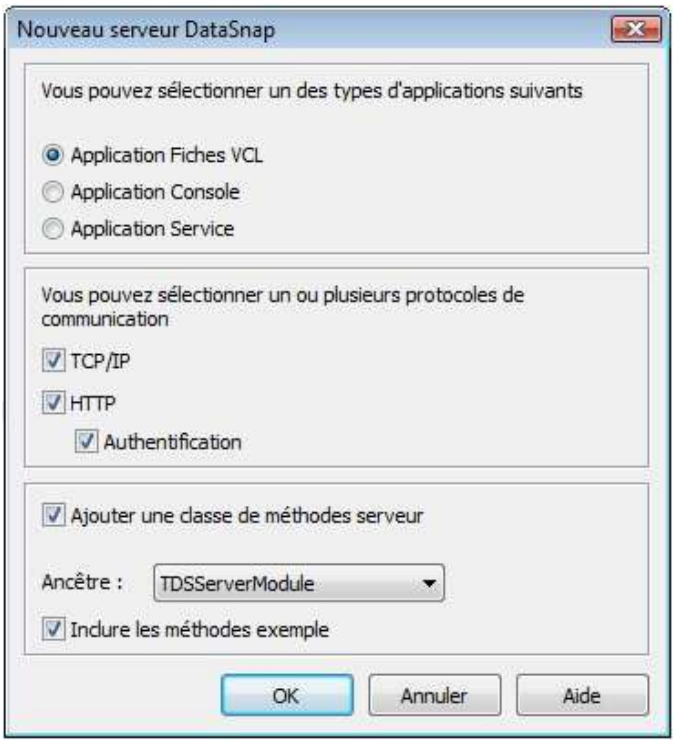
```
TDSServerModuleBase = class(TProviderDataModule)
public
procedure BeforeDestruction; override;
destructor Destroy; override;
end;
{$MethodInfo ON}
TDSServerModule = class(TDSServerModuleBase)
end;
{$MethodInfo OFF}
```

En cas de doute, il convient de sélectionner la classe TDSServerModule.

2.1.1. Groupe de projet multicible – VCL Forms

Comme indiqué, nous allons créer un groupe de projet serveur DataSnap multicible. Nous débuterons par l'application Fiches VCL en sélectionnant tous les protocoles de communication.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



Cette opération crée un nouveau projet – dénommé par défaut Project1.dproj – avec trois unités, dénommées par défaut ServerContainerUnit1.pas, ServerMethodsUnit1.pas et Unit1.pas. Nous devons tout d'abord sélectionner Fichier | Enregistrer le projet sous, pour indiquer des noms de fichiers plus explicites : enregistrer Unit1.pas sous MainForm.pas, ServerContainerUnit1.pas sous ServerContainerUnitDemo.pas, ServerMethodsUnit1.pas sous ServerMethodsUnitDemo.pas et enfin Project1.dproj sous DataSnapServer.dproj. Nous ajouterons dans un instant au groupe de projet l'application Console et l'application Service. Tout d'abord, revenons sur ce que nous venons de créer et tentons de compiler le projet. Si vous compilez le projet DataSnapServer, vous obtiendrez un message d'erreur (car nous avons renommé l'unité générée ServerMethodsUnit1.pas). Le message d'erreur est donc causé par l'unité ServerContainerUnitDemo.pas qui contient l'unité ServerMethodsUnit1 dans la clause d'utilisation de sa section d'implémentation (ligne 30), alors que ServerMethodsUnit1.pas a été enregistré sous ServerMethodsUnitDemo. Pour corriger ce problème, la clause doit être modifiée pour refléter ce changement de nom. Si vous compilez à nouveau, une erreur se produit alors à la ligne 37 où ServerMethodsUnit1 est utilisé pour la classe TServerMethods1. Changez également ici ServerMethodsUnit1 en ServerMethodsUnitDemo pour que le projet Serveur DataSnap puisse être compilé sans encombre.

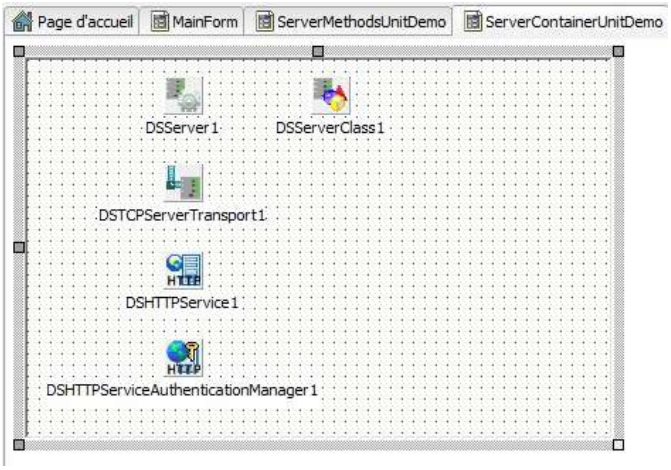
Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

La section d'implémentation de l'unité ServerContainerUnitDemo doit se présenter comme suit:

```
implementation
uses
Windows, ServerMethodsUnitDemo;
{$R *.dfm}
procedure TServerContainer1.DSServerClass1GetClass(
DSServerClass: TDSServerClass; var PersistentClass: TPersistentClass);
begin
PersistentClass := ServerMethodsUnitDemo.TServerMethods1;
end;
end.
```

2.1.1.1. SERVERCONTAINERUNITDEMO

L'onglet Design de l'unité ServerContainerUnitDemo comporte cinq composants : TDSServer, TDSServerClass, TDSTCPServerTransport (communication TCP/IP), TDSHTTPService (communication HTTP) et TDSHTTPServiceAuthenticationManager (authentification HTTP).



Les deux premiers composants sont toujours présents ; les trois autres dépendent naturellement des protocoles de communication optionnels sélectionnés.

2.1.1.1.1. TDSSERVER

Le composant TDSServer n'a que quatre propriétés : AutoStart, HideDSAdmin, Name et Tag. La propriété AutoStart est paramétrée par défaut sur True, ce qui signifie que le Serveur DataSnap démarrera dès que le formulaire est créé. Dans le cas contraire, il est possible d'appeler manuellement la méthode Start – la méthode Stop peut toujours être appelée. La fonction Started permet de déterminer si le Serveur DataSnap a déjà démarré.
Par défaut la propriété HideDSAdmin est paramétrée sur False. Dans le cas contraire (True), les clients se connectant au Serveur DataSnap ne peuvent pas appeler les méthodes serveur intégrées de la classe TDSAdmin. Cette dernière

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

n’est pas à proprement parler une classe mais ses méthodes que nous pouvons appeler sont documentées dans l’unité DSNames :

```
TDSAdminMethods = class
public
const CreateServerClasses = 'DSAdmin.CreateServerClasses';
const CreateServerMethods = 'DSAdmin.CreateServerMethods';
const FindClasses = 'DSAdmin.FindClasses';
const FindMethods = 'DSAdmin.FindMethods';
const FindPackages = 'DSAdmin.FindPackages';
const GetPlatformName = 'DSAdmin.GetPlatformName';
const GetServerClasses = 'DSAdmin.GetServerClasses';
const GetServerMethods = 'DSAdmin.GetServerMethods';
const GetServerMethodParameters = 'DSAdmin.GetServerMethodParameters';
const DropServerClasses = 'DSAdmin.DropServerClasses';
const DropServerMethods = 'DSAdmin.DropServerMethods';
const GetDatabaseConnectionProperties = 'DSAdmin.GetDatabaseConnectionProperties';
end;
```

Le composant TDSServer a cinq événements : OnConnect, OnDisconnect, OnError, OnPrepare et OnTrace. Nous pouvons écrire des gestionnaires pour ces cinq événements répondant aux différentes situations (par exemple, en écrivant une ligne de texte dans un fichier journal).
L’argument des événements OnConnect, OnDisconnect, OnError et OnPrepare est dérivé de TDSEventObject contenant des propriétés pour les composants DxContext, Transport, Server et DbxConnection. Le type TDSConnectEventObject (utilisé pour OnConnect et OnDisconnect) contient également ConnectionProperties et ChannelInfo.
TDSErrorEventObject inclut également l’exception ayant causé l’erreur et TDSPrepareEventObject les propriétés pour MethodAlias et ServerClass que nous souhaitons utiliser.
L’argument du gestionnaire d’événements OnTrace est de type TDBXTraceInfo. Remarquons que ce gestionnaire d’événements généré OnTrace produit également des erreurs – car les types TDBXTraceInfo et CBRType sont inconnus du compilateur. Pour résoudre ce problème, nous devons ajouter l’unité DBXCommon (pour le type TDBXTraceInfo) et l’unité DBCommonTypes (pour CBRType).

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Lors du gestionnaire d'événements OnConnect, nous pouvons examiner ChannelInfo pour la connexion (par exemple en utilisant une fonction spécifique LogInfo pour écrire ces informations dans un fichier journal) :

```
procedure TServerContainer1.DSServer1Connect
(DSConnectEventObject: TDSConnectEventObject);
begin
LogInfo('Connect ' + DSConnectEventObject.ChannelInfo.Info);
end;
```

Et à l'intérieur de OnTrace, nous pouvons journaliser les contenus de TraceInfo.Message pour obtenir une vision claire des opérations réalisées par le serveur.

```
function TServerContainer1.DSServer1Trace(TraceInfo: TDBXTraceInfo): CBRType;
begin
LogInfo('Trace ' + TraceInfo.CustomCategory);
LogInfo(' ' + TraceInfo.Message);
Result := cbrUSEDEF; // action par défaut
end;
```

Remarquons que côté client nous pouvons également tracer la communication entre client et serveur DataSnap en utilisant un composant TSQLMonitor connecté à TSQLConnection (nous démontrerons cette opération plus tard lors de la construction du client pour ce Serveur DataSnap).
Exemple de trace :

```
17:05:55.492 Trace
17:05:55.496 read 136 bytes:{"method":"reader close","params":[1,0]}
   {"method":"prepare","params":[-1,false,"DataSnap.ServerMethod",
   "TServerMethods1.AS GetRecords"]}
17:05:55.499 Prepare
```

Comme vous pouvez le constater, TraceInfo.Message contient des informations sur le nombre d'octets et sur la méthode appelée.

2.1.1.1.2. TDSServerClass

Le composant TDSServerClass est responsable d'indiquer la classe côté serveur – utilisée pour exposer les méthodes publiées aux clients distants (à travers une invocation de méthode dynamique).
La propriété Serveur du composant TDSServerClass pointe sur le composant TDSServer. L'autre propriété importante – en plus de Name et Tag – est LifeCycle paramétrée par défaut sur Session – mais qui peut aussi prendre les valeurs Server ou Invocation. Pour faire bref, Server, Session et Invocation signifient qu'une instance de classe est utilisée pour toute la durée de vie du serveur, pour la durée de la session DataSnap ou simplement pour l'invocation de méthode. Session signifie que chaque connexion entrante obtiendra sa propre instance de la classe serveur. Si vous choisissez Invocation, vous terminerez avec une classe serveur sans-état – utile si vous souhaitez déployer une application de serveur Web CGI (qui est également sans état et chargée/déchargée à chaque requête) ; si vous passez à Server, une seule

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

instance de classe serveur est partagée par toutes les connexions et requêtes entrantes. Ce qui peut être utile (par exemple, si vous souhaitez « compter » le nombre de requêtes) mais il faut s'assurer que cela ne risque pas de causer des problèmes de threading (par exemple, lorsque de multiples requêtes entrantes doivent être traitées simultanément).
Le composant TDSServerClass a quatre événements : OnCreateInstance, OnDestroyInstance (instance créée/détruite), OnGetClass et OnPrepare. Le gestionnaire d'événements OnPrepare peut être utilisé pour préparer une méthode serveur.
En cas d'utilisation de Delphi 2009 ou de Delphi 2010 en plaçant manuellement le composant TDSServerClass dans un conteneur, nous devons implémenter l'événement OnGetClass – car nous devons spécifier quelle classe sera déportée/distanciée du serveur vers le client.
Cependant, les assistants DataSnap de Delphi 2010 implémentent automatiquement le gestionnaire OnGetClass comme suit :

```
procedure TServerContainer1.DSServerClass1GetClass
(DSServerClass: TDSServerClass; var PersistentClass: TPersistentClass);
Begin
PersistentClass := ServerMethodsUnitDemo.TServerMethods1;
end;
```

Remarquons qu'il s'agit du code généré que nous avons légèrement modifié en renommant l'unité ServerMethodsUnit1 pour l'enregistrer sous ServerMethodsUnitDemo.pas.

2.1.1.1.3. TDSTCPSERVERTRANSPORT

Le composant TDSTCPServerTransport est responsable de la communication entre serveur et clients DataSnap en utilisant le protocole TCP/IP.
Le composant TDSTCPserverTransport dispose de cinq propriétés importantes : BufferKBSize, Filters (nouveau dans Delphi 2010), MaxThreads, PoolSize, Port et Server.
La propriété BufferKBSize indique la dimension du buffer de communication (par défaut 32 Ko). La propriété Filters peut contenir une collection de filtres de transport (couverts en détail à la section 4). La propriété MaxThreads permet de définir le nombre maximal de threads (0 par défaut signifiant « pas de limite maximale »). PoolSize peut être mis en œuvre pour activer des pools de connexion et la propriété Port pour contrôler le port TCP/IP utilisé par le serveur pour se connecter au client (si vous modifiez ce paramètre, vous devrez également le modifier par la suite dans le client DataSnap).
La propriété Serveur renvoie au composant TDSServer. Le composant DSTCPServerTransport ne comporte aucun événement.

2.1.1.1.4. TDSHTTPSERVICE

Le composant TDSHTTPService assure la communication entre le serveur DataSnap et les clients DataSnap via le protocole HTTP.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Il a 10 propriétés (en plus de Name et Tag) : Active, AuthenticationManager, DSHostname, DSPort, Filters, HttpPort, Name, RESTContext, Server et ServerSoftware (lecture uniquement).
La propriété Active, qui indique si DSHTTPService écoute les requêtes entrantes, peut être paramétrée sur True lors de la phase de conception ; cela empêchera toutefois l'application serveur DataSnap de démarrer en phase d'exécution (en effet, il ne peut y avoir plus d'un seul composant DSHTTPService actif pour un même port et s'il en existe déjà un au moment de la conception, il est impossible d'en démarrer un second en phase d'exécution). Le meilleur moyen de démarrer TDSHTTPService est de paramétrer la propriété Active sur True dans le gestionnaire d'événements OnCreate de TServerContainer :

```
procedure TServerContainer1.DataModuleCreate(Sender: TObject);
begin
DSHTTPService1.Active := True;
end;
```

La propriété AuthenticationManager est utilisée pour définir le composant gestionnaire prenant en charge l'authentification HTTP ; dans notre exemple, elle renvoie au composant TDSHTTPServiceAuthenticationManager. Celui-ci est détaillé au paragraphe 2.1.1.1.5.
Les propriétés DSHostname et DSPort sont utilisées pour définir le serveur DataSnap auquel se connecter – mais uniquement lorsque la propriété Serveur n'est pas paramétrée. Dans la plupart des cas, il suffit pour y remédier de connecter la propriété Serveur à un composant TDSServer.
La propriété Filters peut contenir un ensemble de filtres de transport, détaillés au paragraphe 4.
La propriété HttpPort est utilisée pour définir le port auquel se connectera le composant DSHTTPService pour écouter les connexions entrantes. Notez que cette valeur de port est fixée par défaut à 80, et doit être modifiée en cas de développement sur une machine équipée d'un serveur Web (par exemple, IIS), puisque ce dernier utilise déjà le port 80 et qu'il est donc impossible d'y connecter le composant TDSHTTPService.
Les tentatives d'exécution de l'application feront apparaître un message d'erreur qui pourra sembler peu compréhensible à première vue. La propriété RESTContext définit l'URL contextuelle REST qui peut être utilisée pour appeler le serveur DataSnap en tant que service REST. Par défaut, RESTContext est paramétré sur « rest », ce qui signifie qu'il est possible d'appeler le serveur http://localhost/datasnap/rest/, comme nous le verrons dans la section 6, portant sur les méthodes de rappel REST, JSON et client.
Enfin, la propriété Serveur devra renvoyer à un composant TDSServer situé dans le même conteneur. Si vous ne souhaitez pas connecter la propriété Serveur, vous pouvez utiliser les propriétés DSHostname et DSPort pour vous connecter au serveur DataSnap Server via TCP. Lorsque la propriété Serveur est paramétrée, les propriétés DSHostname et DSPort sont ignorées.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Le composant TDSHTTPService comporte cinq événements : quatre événements liés à REST et un événement Trace.
Les événements REST sont détaillés au paragraphe 6.
L'événement OnTrace peut être utilisé pour suivre les appels du composant DSHTTPService, comme le montre l'exemple suivant :

```
procedure TServerContainer1.DSHTTPService1Trace(Sender: TObject;
AContext: TDSHTTPContext; ARequest: TDSHTTPRequest;
AResponse: TDSHTTPResponse);
begin
LogInfo('HTTP Trace ' + AContext.ToString);
LogInfo(' ' + ARequest.Document);
LogInfo(' ' + AResponse.ResponseText);
end;
```

Notez que les informations HTTP Trace n'apparaissent que lorsque le client utilise effectivement le protocole HTTP pour se connecter au serveur (et non le protocole TCP/IP par défaut).
Exemple de sortie de trace :

```
17:05:55.398 HTTP Trace TDSHTTPContextIndy
17:05:55.400 /datasnap/tunnel
17:05:55.403 OK
```

Ceci signifie que : AContext est paramétré sur TDSHTTPContextIndy, ARequest est paramétré sur /datasnap/tunnel et AResponse est paramétré sur OK.

2.1.1.1.5. TDSHTTPSERVICEAUTHENTICATIONMANAGER

Le composant TDSHTTPServiceAuthenticationManager sera placé dans le conteneur du serveur si l'option d'authentification du protocole de communication HTTP a été cochée. Bien entendu, il est toujours possible de le placer manuellement dans le conteneur.
Le composant TDSHTTPServiceAuthenticationManager doit être connecté à la propriété AuthenticationManager d'un composant TDSHTTPService.
Il comporte un seul événement, OnHTTPAuthenticate, qui peut être utilisé pour vérifier les informations d'authentification HTTP fournies au Serveur par le Client DataSnap.

```
procedure TServerContainer1.DSHTTPServiceAuthenticationManager1HTTPAuthenticate
(Sender: TObject; const Protocol, Context, User, Password: string;
var valid: Boolean);
begin
if (User = 'Bob') and (Password = 'Swart') then
valid := True
else
valid := False
end;
```

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Bien entendu, cette routine de validation devra être complétée d'une recherche dans la base de données, par exemple à partir d'une version hachée du mot de passe. Pour les envois d'informations utilisateur et mot de passe (haché) du client au serveur, l'authentification HTTP est facilitée par l'emploi du protocole HTTPS sur une partie du transfert ; il est donc à espérer qu'Embarcadero l'ajoutera à la liste actuelle des protocoles HTTP et TCP/IP.
Le protocole HTTPS garantira que la connexion est sécurisée et que le paquet de données est crypté et empêchera tout programme de capter vos données d'utilisateur et mot de passe (hachées). Consultez votre fournisseur de services Internet ou votre Webmestre pour connaître les éventuelles possibilités HTTPS de votre domaine – cette mesure est fortement recommandée (j'utilise par exemple https://www.bobswart.nl).
Une autre technique efficace appliquée par certaines applications serveur DataSnap consiste à écrire les (tentatives) d'authentification HTTP dans un fichier de log pouvant également accueillir les données de Protocole et de Contexte. Ceci peut s'avérer utile pour connaître l'identité des utilisateurs qui s'y connectent et de ceux qui tentent de s'y connecter (par exemple, les fausses tentatives de connexions destinées à accéder au serveur DataSnap).

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

2.1.1.2. SERVERMETHODSUNITDEMO

Après ServerContainerUnitDema.pas, passons maintenant à une autre unité importante de la nouvelle application serveur DataSnap : l'unité ServerMethodsUnitDemo.pas. Dans la boîte de dialogue New DataSnap Server, nous avons spécifié la classe TDSServerModule à utiliser comme ancêtre ; le type TServerMethods1 est donc dérivé de TDSServerModule (qui est dérivé de TDSServerModuleBase, lui-même dérivé de TProviderDataModule, qui ajoute un module de destruction Destroy et une procédure BeforeDestruction). Un TProviderDataModule est dérivé d'un TDataModule normal, et apporte des fonctionnalités supplémentaires de collaboration avec des fournisseurs (ceci sera développé ultérieurement).

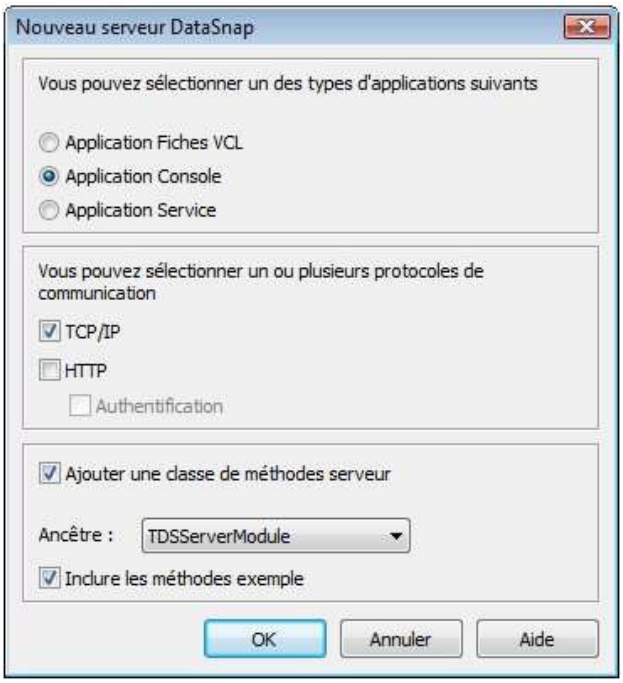
Étant donné que l'une des classes ancêtres de TServerMethods1 est TDataModule, l'onglet de conception indiquera la zone concepteur pour un module de données : les contrôles non-visuels (par exemple, contrôles d'accès aux données) pourront y être placés. Nous verrons le placement des composants d'accès aux données à la section 3 ; dans l'immédiat, nous laisserons la zone de conception vide et continuerons à découvrir comment ajouter des méthodes à la classe TServerMethods1.

Si vous ne souhaitez pas ajouter de cibles supplémentaires au groupe projet pour une application DataSnap Console et/ou DataSnap Windows Service, vous pouvez passer directement à la partie 2.1.4, pour commencer à implémenter les méthodes serveur.

2.1.2. CONSOLE DE GROUPE PROJET MULTI-CIBLE

Retournons maintenant au groupe projet et ajoutons-lui une seconde cible : l'application Console. Cliquez avec le bouton droit sur le nœud du groupe projet et sélectionnez Ajouter un nouveau projet. Dans le référentiel d'objets, allez dans la catégorie Serveur DataSnap et double-cliquez sur l'icône Serveur DataSnap. Sélectionnez cette fois la cible Application Console dans la boîte de dialogue Nouveau serveur DataSnap.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



Notez que quelles que soient les autres options sélectionnées, vous réutiliserez ServerContainerUnitDemo.pas et ServerMethodsUnitDemo.pas à partir du projet DataSnapServer dans tous les cas.

Cliquez sur OK pour générer le nouveau projet. Là encore, ceci aboutira à la création d'un fichier Project1.dproj, ainsi que d'un fichier ServerContainerUnit2.pas et d'un fichier ServerMethodsUnit2.pas. Cliquez avec le bouton droit sur l'unité ServerMethodsUnit2.pas dans le Gestionnaire de projets et retirez-la du nouveau projet.

Conservez ServerContainerUnit2.pas pour l'instant, elle servira à copier une méthode.

Enregistrez le projet dans DataSnapConsoleServer.dproj.

Si vous comparez le contenu de l'unité ServerContainerUnitDemo.pas à celui de la nouvelle unité ServerContainerUnit2.pas (pour l'application console), vous constaterez que cette dernière inclut une procédure globale nommée RunDSServer. Cette procédure globale n'est utile que pour une application Console ; vous devrez donc la copier/coller dans le code source du projet DataSnapConsoleServer.dproj juste avant le début du bloc principal.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

L'outil de contrôle des erreurs signalera alors un certain nombre de problèmes qui peuvent être résolus en ajoutant l'unité Windows à la clause d'utilisation de DataSnapConsoleServer.dpr. DataSnapConsoleServer doit maintenant ressembler à ce qui suit :

```
program DataSnapConsoleServer;
{$APPTYPE CONSOLE}
uses
SysUtils,
Windows,
ServerContainerUnit2 in 'ServerContainerUnit2.pas'
{ServerContainer2: TDataModule};
procedure RunDSServer;
var
LModule: TServerContainer2;
LInputRecord: TInputRecord;
LEvent: DWord;
LHandle: THandle;
begin
Writeln(Format('Starting %s', [TServerContainer2.ClassName]));
LModule := TServerContainer2.Create(nil);
try
LModule.DSServer1.Start;
try
Writeln('Press ESC to stop the server');
LHandle := GetStdHandle(STD INPUT HANDLE);
while True do
begin
Win32Check(ReadConsoleInput(LHandle, LInputRecord, 1, LEvent));
if (LInputRecord.EventType = KEY EVENT) and
LInputRecord.Event.KeyEvent.bKeyDown and
(LInputRecord.Event.KeyEvent.wVirtualKeyCode = VK ESCAPE) then
break;
end;
finally
LModule.DSServer1.Stop;
end;
finally
LModule.Free;
end;
end;
begin
try
RunDSServer;
except
```

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

```
  on E: Exception do
  Writeln(E.ClassName, ': ', E.Message);
  end
  end.
```

Trois nouvelles actions sont maintenant nécessaires et feront malheureusement apparaître de nouveaux problèmes dans l'outil de contrôle des erreurs : le projet DataSnapConsoleServer utilise toujours l'unité ServerContainerUnit2.pas, qui doit donc être retirée. Pour cela, cliquez dessus avec le bouton droit et sélectionnez Retirer du projet. Cliquez ensuite avec le bouton droit sur le nœud DataSnapConsoleServer.exe, sélectionnez Ajouter, choisissez ServerContainerUnitDemo.pas ainsi que ServerMethodsUnitDemo.pas et ajoutez-les au projet.
Toutes les références à TServerContainer2 seront alors signalées comme erreur de syntaxe. ServerMethodsUnitDemo.pas doit définir le type TServerContainer1 ; par conséquent, les problèmes précédemment indiqués peuvent être réglés de la manière suivante : renommez TServerContainer2 dans le code source entre DataSnapConsoleServer et TServerContainer1 (trois emplacements au total). Vous devriez ensuite être en mesure de compiler le nouveau projet DataSnapConsoleServer.dproj et le projet DataSnapServer.dproj original. Vous pourrez alors partager les unités ServerContainerUnitDemo.pas et ServerMethodsUnitDemo.pas entre les deux cibles projets.

2.1.3. GROUPE PROJET MULTI-CIBLE – WINDOWS SERVICE

Après le serveur DataSnap Fiches VCL et le serveur de console DataSnap, une cible reste à ajouter au groupe projet : l'application DataSnap Windows Service. Pour ajouter cette cible, cliquez avec le bouton droit sur le nœud du groupe projet, sélectionnez Ajouter un nouveau projet et double-cliquez à nouveau sur l'icône Serveur DataSnap dans le dialogue Nouveaux éléments. Choisissez Application Service et sélectionnez toutes les options de protocole de communication (TCP/IP, HTTP et HTTP Authentication). Comme nous allons le voir, le conteneur serveur d'une application Service est légèrement différent de celui d'une application Fiches VCL ou Console ; assurez-vous donc de sélectionner, là encore, toutes les options de protocole de communication.
La boîte de dialogue Nouveaux serveur DataSnap affiche alors un nouveau projet, ainsi qu'une unité ServerContainerUnit et une unité ServerMethodsUnit. L'unité ServerMethodsUnit est la même que celle vue précédemment et peut donc être retirée du nouveau projet et remplacée par le fichier ServerMethodsUnitDemo.pas partagé par les applications VCL Forms et Console DataSnap.
En revanche, la nouvelle unité ServerCotainerUnit1.pas est différente : au lieu d'utiliser TDataModule pour placer TDSServer, TDSServerClass et les composants de transport, la classe ainsi obtenue est dérivée de TService (qui contient les composants DataSnap). De plus, quatre méthodes spéciales y ont été ajoutées

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

pour implémenter les événements Stop, Pause, Continue et Interrogate du service :

```
type
TServerContainer3 = class(TService)
DSServer1: TDSServer;
DSTCPServerTransport1: TDSTCPServerTransport;
DSHTTPService1: TDSHTTPService;
DSHTTPServiceAuthenticationManager1: TDSHTTPServiceAuthenticationManager;
DSServerClass1: TDSServerClass;
procedure DSServerClass1GetClass(DSServerClass: TDSServerClass;
var PersistentClass: TPersistentClass);
procedure ServiceStart(Sender: TService; var Started: Boolean);
private
{ Private declarations }
protected
function DoStop: Boolean; override;
function DoPause: Boolean; override;
function DoContinue: Boolean; override;
procedure DoInterrogate; override;
public
function GetServiceController: TServiceController; override;
end;
```

En d'autres termes, l'unité ServerContainerUnitDemo.pas originale ne peut être partagée avec le projet Windows Service, et ServerContainerUnit1.pas doit être renommé ServerContainerUnitServiceDemo.pas. Nous en profiterons, à ce stade, pour enregistrer le projet lui-même dans DataSnapServiceServer.dproj.
Il est donc nécessaire de corriger la référence à l'ancienne unité ServerMethodsUnit2.pas dans ServerContainerUnitServiceDemo en la remplaçant par ServerMethodsUnitDemo.pas pour s'assurer au moins que les mêmes méthodes serveur sont partagées par les trois cibles.

2.1.4. Methodes Serveur

Maintenant que nous disposons d'un ou de plusieurs projets DataSnap Server partageant tous la même unité ServerMethodsUnitDemo, il est temps d'examiner les méthodes serveur plus en détail.
Comme il est mentionné plus haut, la classe TServerMethod1 est l'objet serveur DataSnap qui présente les méthodes (via RTTI) exposées aux clients DataSnap. Si vous avez sélectionné l'option « Inclure les méthodes exemple », une méthode exemple sera déjà présente dans la classe TServerMethods1 : la fonction Echostring. Ajoutons-en une autre pour retourner l'heure actuelle de la machine serveur DataSnap. Pour cela, il suffit de modifier la définition de TServerMethods1 en incluant deux méthodes publiques, comme il est montré ci-dessous :

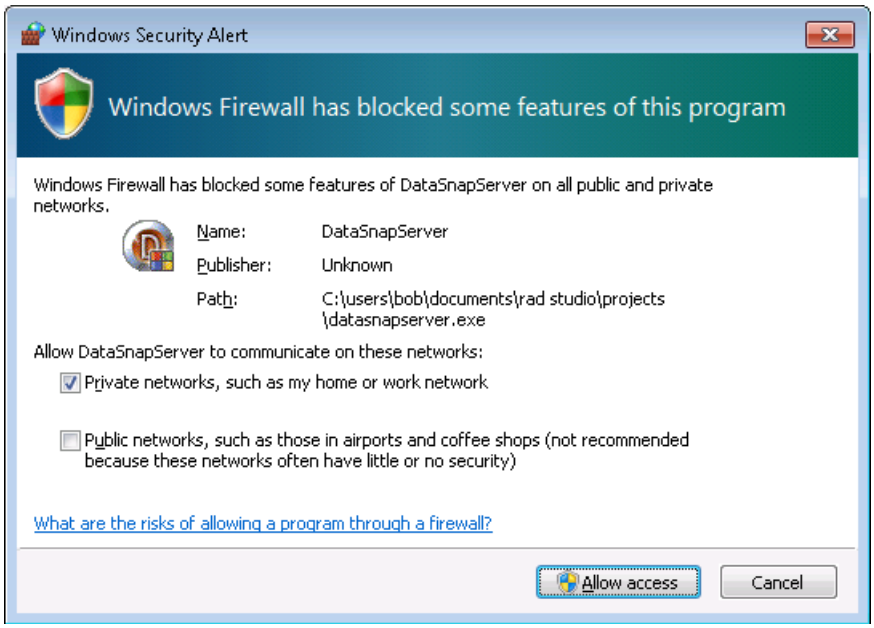
Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

```
type
TServerMethods1 = class(TDSServerModule)
private
{ Private declarations }
public
{ Public declarations }
function EchoString(Value: string): string;
function ServerTime: TDateTime;
end;
```

L'implémentation de la méthode ServerTime ne présente pas de réelle difficulté et est tout aussi rapide que la méthode EchoString de notre exemple :

```
function TServerMethods1.EchoString(Value: string): string;
begin
Result := Value;
end;
function TServerMethods1.ServerTime: TDateTime;
begin
Result := Now
end;
```

Nous pouvons maintenant compiler et exécuter l'application serveur. Si plusieurs cibles projet ont été créées, la plus simple à tester à ce stade est le module exécutable DataSnapServer. En fonction de la version Windows et du niveau des paramètres de sécurité, une alerte de sécurité Windows pourra s'afficher pour signaler que le pare-feu Windows a bloqué certaines fonctionnalités de l'application DataSnapServer.



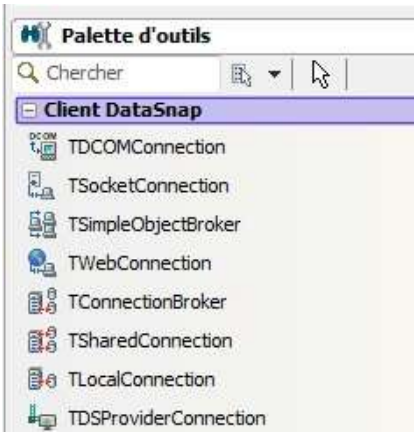
Ceci est dû au fait que l'application DataSnapServer est alors en mode d'écoute active des requêtes entrantes via TCP/IP et HTTP. Elle agira alors comme un

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Cheval de Troie – capable d'écouter les requêtes entrantes ; pour cela, cliquez sur le bouton Autoriser l'accès et poursuivez le démarrage du serveur DataSnap.

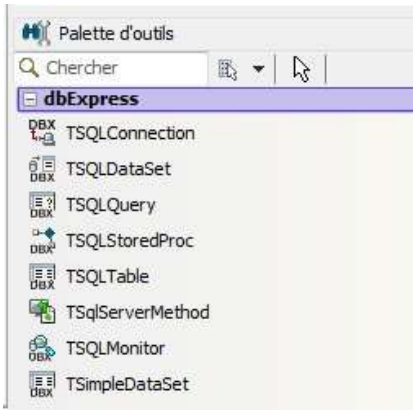
2.2. CLIENT DATASNAP

Une fois que la première démonstration serveur DataSnap est opérationnelle et en mode d'écoute des requêtes entrantes, il est temps de développer un client. Dans cette section, nous verrons comment se connecter au serveur à partir d'un client et comment importer les méthodes en générant des classes serveur. Assurez-vous que le serveur DataSnap est en exécution avant de créer le projet d'application client DataSnap. Pour pouvoir alterner simplement entre les projets serveur DataSnap et le projet client DataSnap en phase de conception, il suffit d'ajouter le projet client DataSnap au même groupe projet. N'importe quel projet peut être un client DataSnap, mais pour cette démonstration, nous utiliserons une application Fiches VCL et l'enregistrerons dans DataSnapClient.dpr (le formulaire sera sauvegardé dans ClientForm.pas). Lorsque la catégorie Serveur DataSnap de la palette d'outils contient les six composants (serveur) DataSnap, la catégorie Client DataSnap ne contient pas les composants que nous devons utiliser à ce stade :



Les composants de la catégorie Client DataSnap sont les « vieux » composants DataSnap, qui sont toujours utilisables ; néanmoins, cette approche n'est pas recommandée pour utiliser DataSnap à ce stade.
À une exception près, toutefois : le nouveau composant TDSProviderConnection, qui peut être utilisé pour connecter un « vieux » serveur DataSnap à un nouveau client (comme nous le verrons au paragraphe 3.2).
Plutôt que la catégorie Client DataSnap, nous examinerons la catégorie dbExpress, contenant un nouveau composant TSQLServerMethod (remarquons sur la capture d'écran que ce nouveau composant est facilement identifiable puisque son préfixe comporte TSql au lieu de TSQL, contrairement aux composants dbExpress existants).

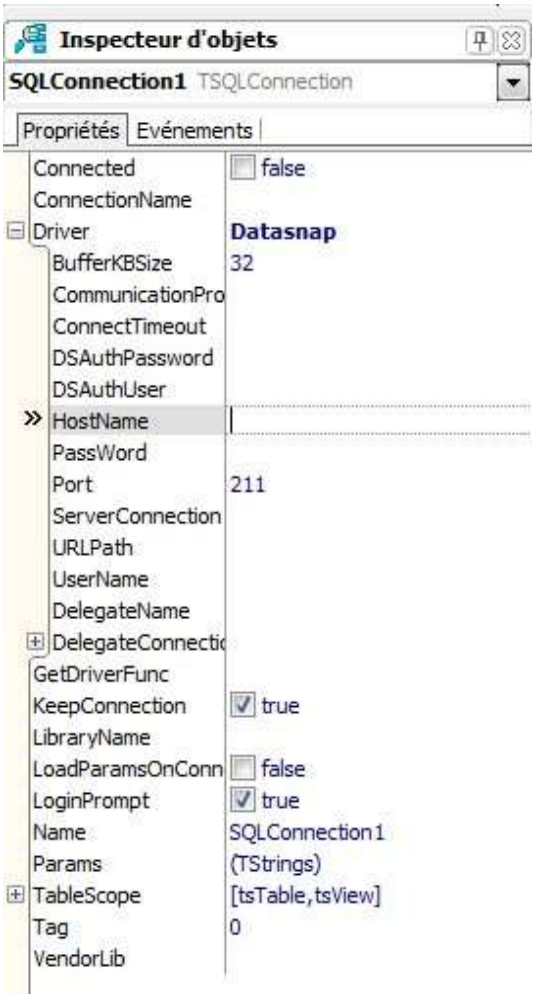
Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



Le composant TSqlServerMethod peut être utilisé pour appeler des méthodes distantes à partir d'un serveur DataSnap, mais doit préalablement être connecté au serveur DataSnap.

Ceci peut être effectué à l'aide d'un composant TSQLConnection – et non avec l'un des anciens composants TxxxConnection. Pour simplifier cette opération, le composant TSQLConnection propose un nouveau nom de pilote dans la liste déroulante (DataSnap).

Il suffit donc de placer un composant TSQLConnection sur le formulaire ClientForm et de paramétrer sa propriété Driver sur DataSnap. La propriété Driver affiche alors un signe « plus » (à gauche du nom de propriété), et peut être développée pour afficher l'ensemble des sous-propriétés.



Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Remarque : si la propriété CommunicationProtocol n'est pas renseignée, le protocole TCP/IP est utilisé par défaut pour la connexion TSQL (et non le port 211 spécifié).
Par défaut, BufferKBSize est paramétré sur 32 (Ko), et le Port sur 211 – exactement comme pour le serveur. En pratique, il est recommandé de choisir un autre numéro de port (aux niveaux serveur et client), car le 211 est communément associé au port assurant la connexion aux serveurs DataSnap.
Le nom d'hôte, nom d'utilisateur et mot de passe sont utilisés pour se connecter au serveur DataSnap. Pour un test local, vous pouvez éventuellement paramétrer HostName sur localhost, mais en règle générale, il est possible de saisir n'importe quel nom d'hôte, nom DNS ou numéro IP direct pour renseigner cette propriété.
Assurez-vous de paramétrer la propriété LoginPrompt du composant TSQLConnection sur False pour éviter que la boîte de dialogue de connexion ne s'affiche.
Après avoir renseigné les propriétés du pilote TSQLConnection, vous pouvez paramétrer la propriété Connected sur True pour (tenter de) vous connecter au serveur DataSnap. Notez que le serveur DataSnap doit être en cours d'exécution au niveau serveur pour réussir cette connexion !

2.2.1. Classes Client DataSnap

Après avoir vérifié que la connexion est possible, cliquez avec le bouton droit sur le composant TSQLConnection et sélectionnez l'option Générer les classes client DataSnap. Cette action générera une nouvelle unité, appelée par défaut Unit1, avec une classe appelée TServerMethods1Client (qui correspond au nom de la classe de méthodes Serveur DataSnap au niveau serveur, complété de la partie « Client »). Enregistrez cette unité dans ServerMethodsClient.pas.
La définition de la classe TServerMethods1Client générée sera la suivante :

```
type
TServerMethods1Client = class
private
FDBXConnection: TDBXConnection;
FInstanceOwner: Boolean;
FEchoStringCommand: TDBXCommand;
FServerTimeCommand: TDBXCommand;
public
constructor Create(ADBXConnection: TDBXConnection); overload;
constructor Create(ADBXConnection: TDBXConnection;
AInstanceOwner: Boolean); overload;
destructor Destroy; override;
function EchoString(Value: string): string;
function ServerTime: TDateTime;
end;
```

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Comme vous pouvez le constater, il existe deux constructeurs pour la classe TServerMethods1Client class et un destructeur ; vous devriez également retrouver les deux méthodes serveur définies au niveau du serveur DataSnap.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Pour utiliser ces méthodes, ajoutez l'unité ServerMethodsClient à la clause d'utilisation du formulaire Client, placez un composant TButton sur le formulaire et écrivez le code suivant dans le gestionnaire d'événements OnClick :

```
procedure TForm2.Button1Click(Sender: TObject);
var
Server: TServerMethods1Client;
begin
Server := TServerMethods1Client.Create(SQLConnection1.DBXConnection);
try
ShowMessage(DateTimeToStr(Server.ServerTime))
finally
Server.Free
end;
end;
```

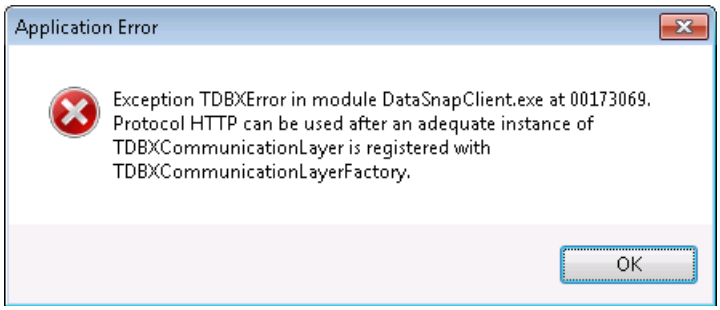
Ce code générera l'instance de TServerMethods1Client ; appelez ensuite la méthode serveur ServerTime, puis supprimez à nouveau le proxy du serveur DataSnap.
En cliquant sur ce bouton, un message s'affichera pour indiquer la valeur date et heure du serveur, comme prévu.

Le lecteur est invité à s'exercer en appliquant cette même procédure pour tester la méthode EchoString.

2.2.1.1. Protocole de Communication HTTP

Comme nous l'avons mentionné, TCP/IP est le protocole de communication par défaut du composant TSQLConnection. Ceci signifie également que nous ne retrouvons aucun des messages HTTP Trace (définis au point 2.1.1.1.4). Notez toutefois que pour changer de protocole de communication, il suffit d'indiquer HTTP en tant que valeur de la sous-propriété CommunicationProtocol de la propriété Pilote de TSQLConnection.
Ceci implique également que la propriété de Port doit être modifiée, puisque le port 211 était utilisé pour TCP/IP. Assurez-vous de spécifier pour le Port la même valeur que celle choisie pour le composant TDSHTTPService dans le Conteneur serveur.
Après avoir procédé à ces changements et réexécuté le Client DataSnap, vous verrez probablement s'afficher le message d'erreur suivant :

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

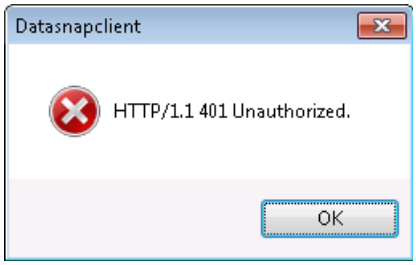


Pour y remédier, il suffit d'ajouter l'unité DSHTTPLayer à la clause d'utilisations (de ClientForm par exemple) du Client DataSnap.

2.2.1.2. AUTHENTIFICATION HTTP

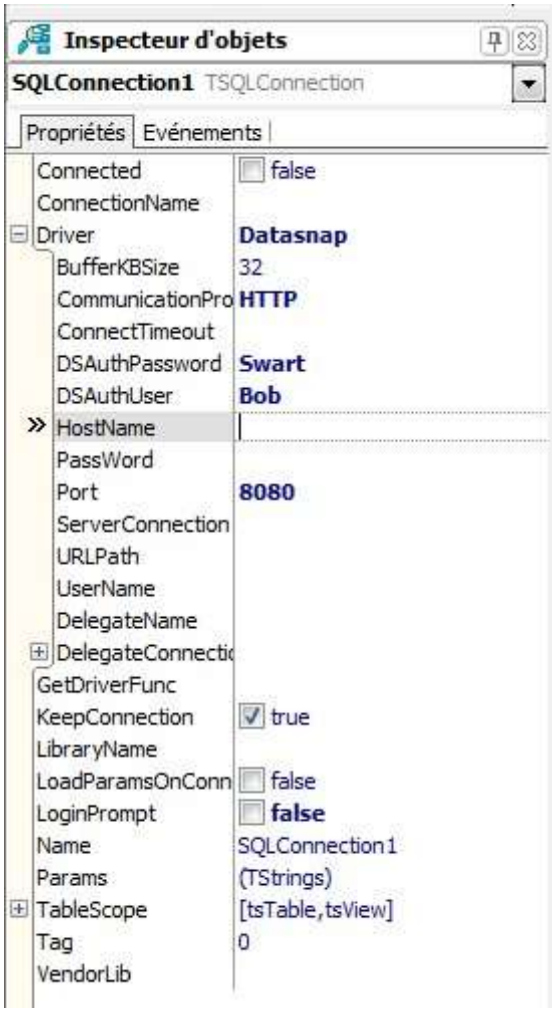
L'utilisation du protocole de communication HTTP présente notamment l'avantage de pouvoir inclure l'Authentification HTTP. Cette fonctionnalité est supportée au niveau du Serveur DataSnap par le composant TDSHTTPServiceAuthenticationManager, comme il est expliqué au point 2.1.1.1.5.

Dès que le gestionnaire d'événements OnHTTPAuthenticate est implémenté et qu'une vérification d'authentification HTTP y est effectuée, vous devez vérifier si les bonnes informations sont transmises, pour obtenir l'accès de TDSHTTPServiceAuthenticationManager. Dans le cas contraire, un message d'accès non-autorisé HTTP/1.1 401 s'affichera :



Pour transmettre le nom d'utilisateur et le mot de passe d'authentification HTTP du Client au Serveur DataSnap, et plus précisément au composant TDSHTTPServiceAuthentication, il est nécessaire de renseigner les propriétés DSAuthUser et DSAuthPassword du composant TSQLConnection (sur le formulaire client).

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



Notez que vous devez également indiquer une valeur pour le nom d'hôte (HostName), sauf si vous réalisez des tests avec un Serveur DataSnap sur la même machine locale.

2.3. DÉPLOIEMENT SERVEUR DATASNAP

L'exemple fonctionne bien lorsque le serveur et le client sont tous deux exécutés sur la même machine locale ; or, en pratique, le Serveur DataSnap sera exécuté sur une machine serveur, avec un ou plusieurs clients connectés à cette machine en réseau. Le plus souvent, la machine sur laquelle est déployée l'application serveur DataSnap ne dispose pas d'une installation Delphi. Ceci implique qu'il vous faudra peut-être envisager de compiler le serveur DataSnap sans disposer de solutions d'exécution activées, et que vous obtiendrez un seul gros module exécutable.

Puisqu'aucun composant d'accès aux données n'a été utilisé, aucun pilote de bases de données supplémentaire ni DLL externe n'est nécessaire à ce stade.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

2.3.1. Déploiement Client DataSnap

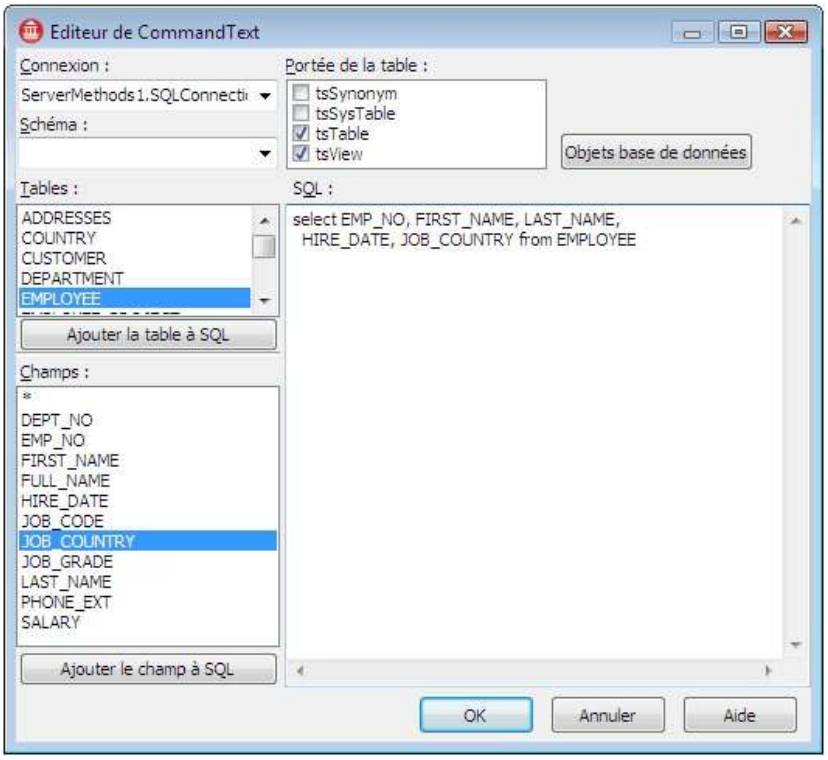
Dans l'hypothèse où l'application client DataSnap est exécutée sur une autre machine que l'application serveur, vous devrez vous assurer que le client peut se connecter au serveur. Pour cela, le composant TSQLConnection du formulaire client devra spécifier non seulement les valeurs des sous-propriétés CommunicationProtocol et Port de la propriété Driver, mais également celle de HostName. Évitez d'affecter une adresse IP et utilisez plutôt un nom DNS logique, si possible. Il est possible, par exemple, d'utiliser la valeur HostName www.bobswart.nl (il n'est pas nécessaire de spécifier le préfixe http:// puisque le protocole effectivement utilisé sera spécifié dans le protocole de communication).

3. DataSnap et bases de données – ou vous voulez

En plus du développement de méthodes serveur simples avec le framework Delphi 2010 DataSnap, il est possible d'ajouter au serveur l'accès à la base de données, en transformant l'architecture en application de base de données multiniveau dans laquelle le serveur DataSnap se connectera à la base de données et les clients DataSnap seront « légers » ou transparents, sans nécessiter de pilote de base de données (du côté client).
Dans le déploiement DataSnap réalisé jusqu'à présent, nous nous sommes contentés d'utiliser directement le composant SQLConnection et les classes client générées, mais il est également possible d'utiliser les deux nouveaux composants DataSnap TsqlServerMethod et TDSProviderConnection.
Assurez-vous, pour commencer, que le serveur fournit effectivement un ensemble TDataSet ; pour cela, retournez à ServerMethodsUnitDemo et placez un composant TSQLConnection sur le module de données.
Connectez le composant TSQLConnection à votre SGBD et votre table favoris (dans l'exemple, la table Employees de la base de données BlackfishSQL exemple).
Pour cela, placez un composant TSQLConnection sur la zone du concepteur ServerMethods1 (dans l'unité ServerMethodsUnitDemo.pas). Paramétrez la propriété Driver du composant TSQLConnection sur BlackfishSQL. Ouvrez ensuite la propriété Driver et paramétrez la propriété Database sur le chemin vers employee.jds dans C:\Documents and Settings\All Users\Documents\RAD Studio\7.0\Demos\database\databases\BlackfishSQL sous Windows XP ou dans C:\Users\Public\Documents\ RAD Studio\7.0\Demos\database\databases\BlackfishSQL sous Windows Vista ou Windows 7.
Assurez-vous que la propriété LoginPrompt du composant TSQLConnection est paramétrée sur False et ensuite tentez de paramétrer la propriété Connected sur True pour vérifier si vous pouvez ouvrir la connexion à la base de données Employee.jds.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Placez ensuite un composant TSQLDataSet à côté de TSQLConnection et connectez sa propriété SQLConnection au composant TSQLConnection.



Laissez ensuite le jeu CommandType sur ctQuery et faites cliquez sur la propriété ellipse pour que la propriété CommandText entre une requête SQL.

```
SELECT EMP_NO, FIRST_NAME, LAST_NAME, HIRE_DATE, JOB_COUNTRY FROM EMPLOYEE
```

Assurez-vous que les propriétés LoginPrompt et Connected du composant TSQLConnection sont sur False ainsi que la propriété Active de TSQLDataSet. Nous devons maintenant ajouter une fonction publique à la classe TServerMethods1 de l'unité ServerMethodsUnitDemo pour retourner les contenus du composant TSQLDataSet.

```
type
TServerMethods1 = class(TDSServerModule)
SQLConnection1: TSQLConnection;
SQLDataSet1: TSQLDataSet;
private
{ Private declarations }
public
{ Public declarations }
function EchoString(Value: string): string;
function ServerTime: TDateTime;
function GetEmployees: TDataSet;
end;
```

Comme vous pouvez le voir dans la définition TServerMethod1 la nouvelle fonction est dénommée GetEmployees et l'implémentation se présente comme suit :

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

```
function TServerMethods1.GetEmployees: TDataSet;
begin
SQLDataSet1.Open; // make sure data can be retrieved
    Result: = SQLDataSet1
end;
```

Recompilez le serveur et assurez-vous qu'il fonctionne encore. Remarquez que si vous avez fermé le serveur DataSnap mais ne pouvez pas le recompiler, il est probable que le projet serveur DataSnap est toujours en cours d'exécution.

3.1. TSQLSERVERMETHOD

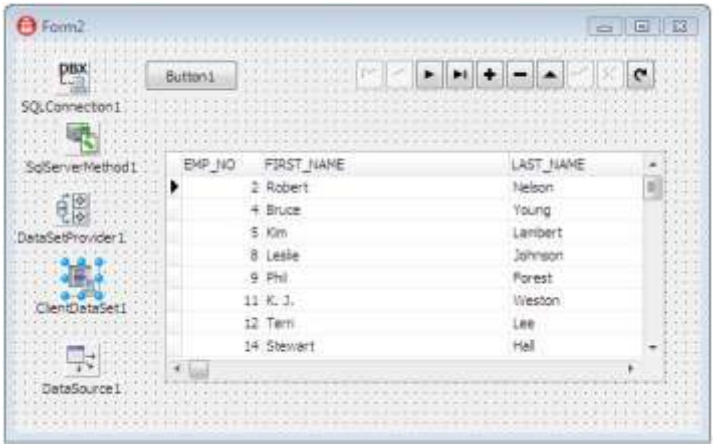
Retournez à l'application client DataSnap. Le composant TSQLConnection ne doit alors plus être connecté au serveur DataSnap (dans le cas contraire, il peut s'agir de la raison pour laquelle vous n'êtes pas parvenu à recompiler le serveur DataSnap dans la mesure où le processus est toujours en cours d'utilisation). Nous pouvons paramétrer à nouveau Connected sur True pour rafraîchir l'information depuis le serveur et ensuite régénérer les classes client (avec le bouton droit).
Pour réécrire l'unité ServerMethodsClient précédente, il est recommandé de supprimer la version précédente du projet et de sélectionner à nouveau « Générer les classes client DataSnap ». Si nous sauvegardons la nouvelle unité générée dans le fichier ServerMethodsClient.pas, nous n'aurons pas à modifier le code client précédent. Après avoir exécuté à nouveau la tâche « Générer les classes client DataSnap », la classe TServerMethods1Client générée a été étendue avec la méthode GetEmployees (listant également les fonctions ServerTime et EchoString qui sont toujours disponibles) :

```
type
TServerMethods1Client = class
private
FDBXConnection: TDBXConnection;
FInstanceOwner: Boolean;
FEchoStringCommand: TDBXCommand;
FServerTimeCommand: TDBXCommand;
FGetEmployeesCommand: TDBXCommand;
public
constructor Create(ADBXConnection: TDBXConnection); overload;
constructor Create(ADBXConnection: TDBXConnection;
AInstanceOwner: Boolean); overload;
destructor Destroy; override;
function EchoString(Value: string): string;
function ServerTime: TDateTime;
function GetEmployees: TDataSet;
end;
```

Pour utiliser la méthode GetEmployees pour récupérer le jeu TDataSet, nous pouvons utiliser un composant TsqlServerMethod de la catégorie dbExpress dans

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

la Palette d'outils. Placez TsqlServerMethod sur le formulaire client, connectez sa propriété SQLConnection sur SQLConnection1 puis ouvrez la liste déroulante ServerMethodName pour afficher toutes les méthodes disponibles que nous pouvons appeler : un certain nombre de méthodes DSAdmin (qui peuvent être affichées en paramétrant la propriété HideDSAdmin du composant TDSServer sur True), suivies par 3 méthodes DSMetadata, 7 méthodes TServerMethods.AS_xxx (interface IAppServer originale) et finalement nos méthodes TServerMethods1 EchoString, ServerTime et GetEmployees.
Dans l'exemple actuel, nous devons sélectionner TServerMethods1.GetEmployees comme valeur de la propriété ServerMethodName. Le résultat de SqlServerMethod est alors un jeu de données contenant les enregistrements employés (ou au moins avec le résultat de notre déclaration SQL).
Nous pouvons utiliser maintenant la chaîne habituelle TDataSetProvider, TClientDataSet et TDataSource pour afficher les données dans une grille TDBGrid.
Placez TDataSetProvider dans le formulaire client et connectez sa propriété DataSet sur le composant SqlServerMethod. Placez ensuite un composant TClientDataSet dans le formulaire client et connectez sa propriété ProviderName sur le composant DataSetProvider. Laissez la propriété RemoteServer en blanc – utilisé uniquement dans l'ancienne approche DataSnap mais pas dans la nouvelle architecture DataSnap.
Finallement, placez TDataSource dans le formulaire client et connectez sa propriété DataSet au composant TClientDataSet. Nous pouvons ensuite utiliser TDBGrid et TDBNavigator pour connecter leur propriété DataSource au composant TDataSource pour afficher les données dans le formulaire client.
Pour vérifier la connexion en phase de conception, nous pouvons paramétrer la propriété Active de ClientDataSet sur True. Ceci fait basculer la propriété Active de SqlServerMethod (seulement pour un instant pour récupérer les données, après quoi la propriété Active revient sur False), ce qui définit la propriété Connected du composant TSQLConnection sur True afin de permettre la connexion entre client et serveur DataSnap.
La connexion demeure active moyennant quoi, à la fois TClientDataSet et TSQLConnection, sont actifs et affichent les données dans TDBGrid :



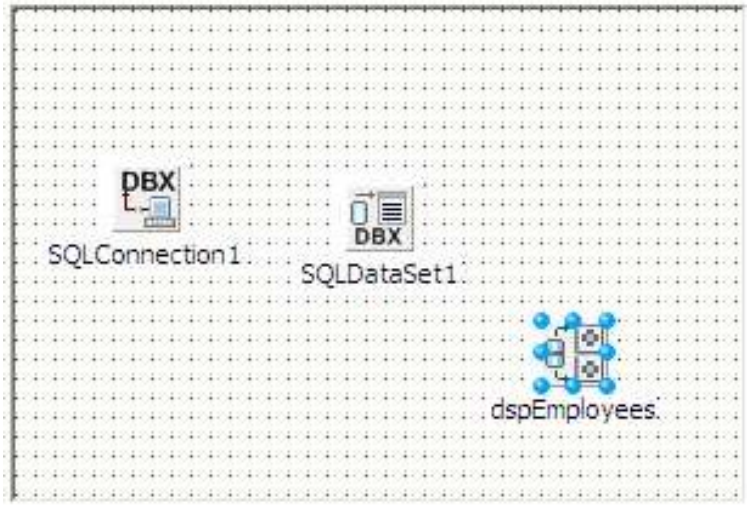
Ceci fournit un moyen simple d'afficher les données en mode lecture seule (dans la mesure où TSQLServerMethod n'autorise pas la combinaison

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

TDataSetProvider-TClientDataSet à envoyer des mises à jour du client au serveur DataSnap).
TSQLServerMethod est une solution légère et pratique pour se connecter à des données en lecture seule.
Par conséquent, si vous souhaitez fournir des données d'un serveur DataSnap à des personnes ne devant en aucun cas pouvoir les modifier, l'architecture actuelle, exposant les résultats TSQLDataSet de la classe TServerMethods1, est une bonne idée.
Cependant, si des mises à jour sont nécessaires nous devons utiliser une autre approche pour exposer les données.

3.2. TDSProviderConnection

Si nous devons appliquer des mises à jour, nous avons besoin d'un composant TDSProviderConnection qui nous donne une référence à un DataSetProvider du côté serveur permettant non seulement de lire les données mais aussi d'envoyer les mises à jour.
Nous devons tout d'abord effectuer un changement sur le module de données du serveur. Cependant, nous n'avons pour l'instant fait qu'ajouter une fonction pour retourner TDataSet et nous devons ajouter maintenant un véritable TDataSetProvider et nous assurer qu'il est exporté du serveur vers les clients DataSnap. Par conséquent, retournons à l'unité ServerMethodsUnitDemo et plaçons un composant TDataSetProvider à côté de TSQLDataSet, pointant vers la propriété DataSet de TDataSetProvider vers TSQLDataSet. Nous devons également renommer TDataSetProvider pour le rendre intelligible (dspEmployees).



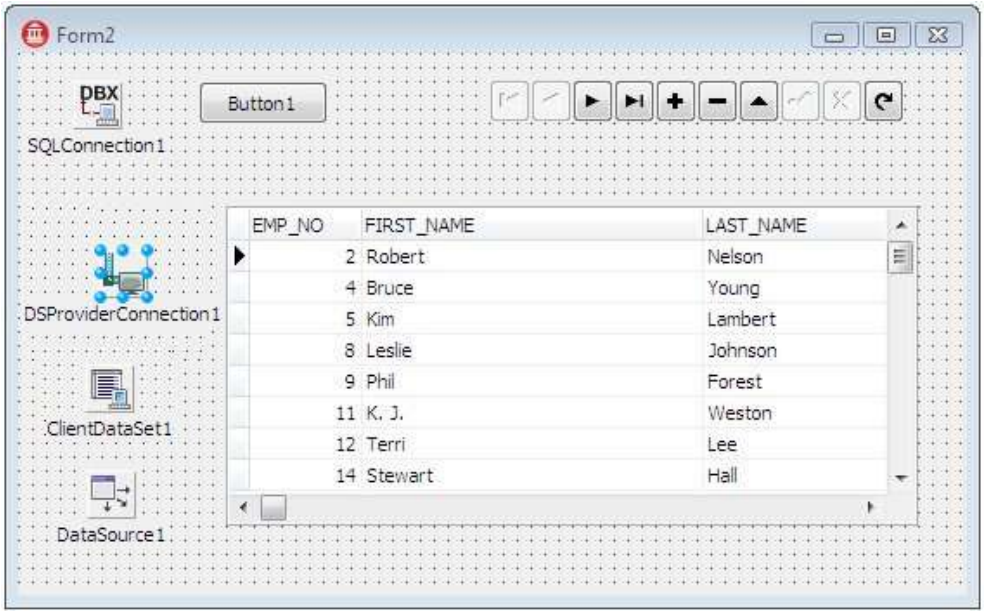
Recompilons et exécutons à nouveau le serveur DataSnap pour modifier le client et appliquer des changements aux données exposées.

3.2.1. Client TDSProviderConnection

Pour modifier le client DataSnap afin d'utiliser le composant exposé TDataSetProvider, nous devons supprimer les composants TSQLServerMethod et

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

TDataSetProvider du formulaire client et ajouter à la place un composant TDSProviderConnection.
Pointez la propriété SQLConnection de TDSProviderConnection sur le composant TSQLConnection se connectant au serveur DataSnap. Nous devons également entrer une valeur pour la propriété ServerClassName de TDSProviderConnection. Malheureusement, nous ne disposons pas encore d'une liste déroulante à ce niveau. Nous devons maintenant entrer manuellement le nom de TDSServerModule (dans notre cas : TServerMethods1).
Dans l'exemple précédent, TClientDataSet ne connecte que son ProviderName à DataSetProvider1. Cependant, en utilisant le composant TDSProviderConnection, nous devons tout d'abord affecter la propriété RemoteServer de TClientDataSet au composant TDSProviderConnection puis sélectionner une nouvelle valeur pour la propriété ProviderName (qui était toujours affectée à la valeur DataSetProvider1, et doit désormais pointer vers dspEmployees – le nom plus explicite du composant TDataSetProvider exposé par le serveur DataSnap).
La liste déroulante de la propriété ProviderName doit désormais proposer l'option dspEmployees (nom du composant TDataSetProvider exporté de l'unité ServerDataMod).
Nous pouvons maintenant définir à nouveau la propriété Active du composant TClientDataSet sur True pour afficher les données réelles en conception :



Cette opération a également pour effet de paramétrer la propriété Connected du composant TSQLConnection sur True. Remarquons, qu'il ne serait pas bon de laisser ces deux propriétés sur True dans le formulaire client en conception.
Avant tout, si vous ouvrez le projet Client DataSnap dans Delphi, il tentera de se connecter au serveur DataSnap ce qui provoquera une erreur si le serveur n'est pas démarré. Ensuite, si vous démarrez l'application en exécution et qu'aucune connexion n'est disponible, l'application générera également une exception. Ceci vous empêche d'utiliser l'application avec des données locales ; elle est donc inopérante en l'absence de connexion.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Le meilleur moyen consiste donc à inclure une option de menu ou un bouton de connexion explicite au composant TSQLConnection et/ou à activer TClientDataSet. C'est également le bon moment pour inclure les informations nom utilisateur/mot de passe (nous y reviendrons). Pour l'instant, assurez-vous que la propriété Active de TClientDataSet est sur False ainsi que la propriété Connected du composant TSQLConnection. Ensuite, placez un bouton sur le formulaire client et ouvrez explicitement TClientDataSet dans le gestionnaire d'événements OnClick.

```
procedure TForm2.Button2Click(Sender: TObject);
begin
ClientDataSet1.Open;
end;
```

Il est maintenant temps d'ajouter du code pour véritablement changer les données (côté client) et de retourner les modifications au serveur.

3.2.2 MISES A JOUR DE LA BASE DE DONNÉES

Il existe deux méthodes pour retourner au serveur les modifications réalisées sur le client : automatique ou manuel. Les deux font appel à la fin à la même méthode mais l'invocation est soit automatique, soit commandée par l'utilisateur (chacune ayant des avantages et des inconvénients).
Dans l'approche automatique, nous pouvons utiliser les événements OnAfterInsert, OnAfterPost et OnAfterDelete de TClientDataSet, puisqu'il s'agit des méthodes ayant apporté des changements aux données. Dans les gestionnaires d'événements – qui peuvent être partagés par la même implémentation – nous pouvons appeler la méthode ApplyUpdates de TClientDataSet pour envoyer les changements (ou « Delta ») au serveur pour retour à la base de données.

```
procedure TForm2.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
ClientDataSet1.ApplyUpdates(0);
end;
```

Si une erreur se produit (enregistrement non trouvé) lors de la mise à jour, nous pouvons obtenir un retour dans l'événement OnReconcileError de TClientDataSet (voir les détails à la section 3.2.3).
L'envoi manuel des mises à jour au serveur DataSnap utilise également la méthode ApplyUpdates de TClientDataSet mais ne doit cette fois pas être appelée par les gestionnaires d'événement OnAfterInsert, OnAfterPost et OnAfterDelete. Au lieu de cela, nous devons ajouter un bouton au formulaire client pour permettre à l'utilisateur de retourner explicitement les modifications au serveur.

```
procedure TForm2.btnUpdateClick(Sender: TObject);
ClientDataSet1.ApplyUpdates(0);
end;
```

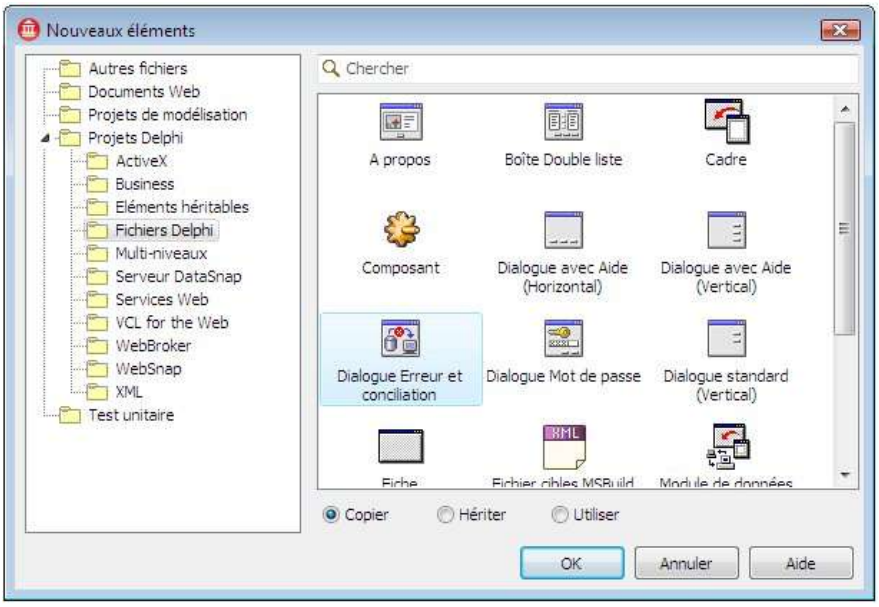
Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

L'avantage de l'appel automatique à ApplyUpdates est naturellement que l'utilisateur ne pourra pas oublier de retourner les changements au serveur. Cependant, le désavantage reste qu'il est impossible d'annuler : lorsqu'elles sont postées, les données sont appliquées au serveur. D'un autre côté, en cas d'approche manuelle, tous les changements sont conservés côté client dans la mémoire du composant TClientDataSet. Ceci permet à l'utilisateur d'annuler certains changements (le dernier, un enregistrement spécifique ou l'ensemble des mises à jour en attente). Cliquer sur le bouton de mise à jour pour appeler explicitement les méthodes ApplyUpdates lorsque l'utilisateur est prêt présente un risque d'oubli, nous devons par conséquent ajouter une vérification au formulaire ou à l'application pour l'empêcher de se fermer s'il reste des modifications dans TClientDataSet. Ce dernier peut être vérifié ultérieurement en recherchant dans la propriété ChangeCount de TClientDataSet.

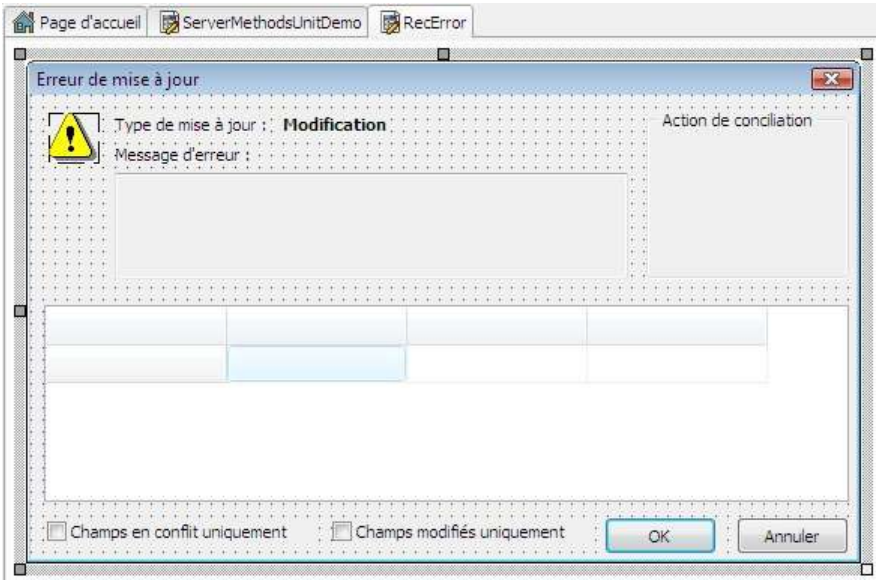
3.2.3. RECONCILIATION DES ERREURS

La méthode ApplyUpdates du composant TClientDataSet a un argument : le nombre maximum d'erreurs autorisées avant l'arrêt de l'application d'autres mises à jour. Si deux clients se connectent au serveur DataSnap, obtiennent les données Employees et réalisent tous deux des changements sur le premier enregistrement, d'après ce que nous avons construit jusque-là, les deux clients pourraient retourner l'enregistrement mis à jour au serveur DataSnap en utilisant la méthode ApplyUpdates de leur composant TClientDataSet. Si les deux passent la valeur zéro comme argument MaxErrors d'ApplyUpdates alors le second qui tentera d'effectuer la mise à jour sera stoppé. Le deuxième client pourrait passer une valeur numérique supérieure à zéro pour indiquer qu'un nombre fixe d'erreurs/conflits est autorisé avant l'arrêt de la mise à jour. Cependant, même si le second client passe -1 comme argument (pour indiquer que la mise à jour doit se poursuivre quel que soit le nombre d'erreurs), les enregistrements modifiés par le précédent client ne seront pas mis à jour. En d'autres termes, des actions de réconciliations doivent être effectuées pour gérer les mises à jour des enregistrements et champs déjà modifiés.
Heureusement, Delphi propose une fonction utile conçue spécifiquement pour ce cas. Chaque fois qu'il est nécessaire de réconcilier des erreurs, il est préférable de l'ajouter à l'application Client DataSnap (ou d'en écrire une).
Pour utiliser celle de Delphi, sélectionnez *Fichier* | *Nouveau* - *Autre*, allez dans la sous-catégorie Fichiers Delphi de Projets Delphi dans le dialogue Nouveaux éléments et sélectionnez l'icône Dialogue Erreur et conciliation.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



Cliquez ensuite sur OK, pour ajouter RecError.pas à votre projet DataSnapClient. Cette unité contient la définition et l'implémentation du dialogue Erreur de mise à jour utilisable pour résoudre les erreurs de mise à jour de la base de données.



Une instance de ReconcileErrorForm est créée dynamiquement, à la volée lorsque cela est nécessaire. La question qui se pose alors est de savoir *quand l'utiliser ou ne pas l'utiliser.* La réponse est en fait simple, pour chaque enregistrement dont la mise à jour a échoué (quelle qu'en soit la raison), le gestionnaire d'événement OnReconcileError du composant TClientDataSet est appelé ; il est défini comme suit :

```
procedure TForm2.ClientDataSet1ReconcileError(DataSet: TClientDataSet;
E: EReconcileError; UpdateKind: TUpdateKind;
var Action: TReconcileAction);
```

Ce gestionnaire dispose de quatre arguments : le composant TClientDataSet ayant produit l'erreur ; un ReconcileError spécifique contenant un message sur la cause de la condition d'erreur ; le type UpdateKind (insérer, effacer ou modifier) ayant généré l'erreur et enfin l'Action qui doit être réalisée.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Le paramètre Action peut retourner les valeurs suivantes (dans l'ordre des valeurs enum réelles) :

- raSkip – Ne pas mettre à jour l'enregistrement mais conserver les modifications non réalisées dans le log de changement. Prêt pour un nouvel essai la prochaine fois.
- raAbort – Annulation de l'ensemble de la réconciliation ; aucun autre enregistrement ne sera passé au gestionnaire d'événements OnReconcileError.
- raMerge – Fusion de l'enregistrement mis à jour avec l'enregistrement actuel de la base de données (distante), modification uniquement des valeurs de champ (distants) si elles ont changé de votre côté.
- raCorrect – Remplacement de l'enregistrement mis à jour par une valeur corrigée de l'enregistrement du gestionnaire d'événement (ou dans ReconcileErrorDialog). Avec cette option l'intervention utilisateur est requise (saisie).
- raCancel – Annulation de tous les changements à l'intérieur de l'enregistrement et retour à l'enregistrement (local) original.
- raRefresh – Annulation de tous les changements à l'intérieur de l'enregistrement avec rechargement des valeurs de la base de données (distante) actuelle (et non depuis l'enregistrement local original).

L'intérêt de ReconcileErrorForm est de s'affranchir de ces considérations. Il suffit d'inclure l'unité ErrorDialog dans la définition de formulaire principal du client DataSnap, de cliquer sur le formulaire Client DataSnap et de choisir *Fichier | Utiliser l'unité* pour obtenir le dialogue Utiliser l'unité. Avec le formulaire client comme unité courante, le dialogue Utiliser l'unité liste la seule unité disponible : ErrorDialog.
Il suffit de la sélectionner et de cliquer sur OK.
La seconde opération à réaliser est d'écrire une ligne de code dans le gestionnaire d'événements OnReconcileError pour appeler la fonction HandleReconcileError de l'unité ErrorDialog (qui vient d'être ajoutée à la liste d'importation ClientMainForm). La fonction HandleReconcileError dispose des mêmes quatre arguments que OnReconcileError (ce qui n'est naturellement pas une coïncidence). Il s'agit donc de passer les arguments de l'un à l'autre – rien de plus, rien de moins. Le gestionnaire OnReconcileError du composant TClientDataSet peut être codé comme suit :

```
procedure TFrmClient.ClientDataSet1ReconcileError(DataSet: TClientDataSet;
E: EReconcileError; UpdateKind: TUpdateKind;
var Action: TReconcileAction);
begin
Action := HandleReconcileError(DataSet, UpdateKind, E)
end;
```

3.2.4. DEMONSTRATION DE LA RECONCILIATION DES ERREURS

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

La question est désormais de savoir comment cela fonctionne en pratique. Pour le tester, vous devez naturellement disposer de deux (ou plus) clients DataSnap fonctionnant simultanément. Pour un test complet utilisant le client DataSnap actuel et les applications serveur DataSnap, vous devez passer par les étapes suivantes :

- Démarrez l'application serveur DataSnap.
- Démarrez le premier client DataSnap et cliquez sur le bouton Connect.
- Démarrez le second client DataSnap et cliquez sur le bouton Connect. Les données seront obtenues du même Serveur DataSnap qui fonctionne déjà.
- Utilisez le premier client DataSnap pour modifier le champ "FirstName" du premier enregistrement.
- Utilisez le second client DataSnap pour modifier également le champ "FirstName" du premier enregistrement.
- Cliquez sur le bouton "Valider modifications" du premier Client DataSnap. Toutes les mises à jour seront appliquées sans problème.
- Cliquez sur le bouton "Valider modifications" du second Client DataSnap. Cette fois une ou plusieurs erreurs se produisent car la valeur du champ "FirstName" du premier enregistrement a été modifiée (par le premier client DataSnap) ; pour cela et éventuellement pour d'autres enregistrements en conflit le gestionnaire d'événements OnReconcileError est appelé.
- Dans le dialogue Erreur de mise à jour, vous pouvez désormais expérimenter les Actions de réconciliation (passer, annuler, fusionner, corriger, annuler et rafraîchir) pour mieux comprendre leurs effets. Notez en particulier les différences entre Passer et Annuler et entre Corriger, Rafraîchir et Fusionner.

Passer amène à l'enregistrement suivant en sautant la mise à jour requise (pour le moment). Le changement non-appliqué demeure dans le log de changement. *Annuler* saute également la mise à jour requise mais annule toutes les mises à jour ultérieures (du même lot de mises à jour). La requête de mise à jour en cours est sautée dans les deux cas mais avec Passer les autres requêtes de mise à jour se poursuivent alors qu'Annuler annule l'ensemble de la requête ApplyUpdates.
Rafraîchir oublie toutes les mises à jour effectuées à l'enregistrement et le rafraîchit avec la valeur actuelle de la base de données du serveur. *Fusionner* tente de fusionner l'enregistrement mis à jour avec celui du serveur et place vos changements à l'intérieur de l'enregistrement serveur. Rafraîchir et Fusionner ne poursuivent pas la requête de changement, les enregistrements sont synchronisés après Rafraîchir et Fusionner (la requête de changement peut être ré-exécutée après Passer ou Annuler).
Corriger, l'option la plus puissante, permet de personnaliser la mise à jour dans le gestionnaire d'événements. Pour cela, il est nécessaire d'écrire du code ou de saisir les valeurs vous-même.

3.3. DÉPLOIEMENT BASE DE DONNÉES DATASNAP

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Le déploiement d'un serveur DataSnap utilisant des bases de données peut être plus complexe que le déploiement du serveur DataSnap avec lequel nous avons débuté. Pour l'application client, rien ne change – il s'agit toujours d'un client léger/intelligent qui peut être déployé comme un exécutable autonome en ajoutant l'unité MidasLib aux clauses d'utilisation.
Pour le serveur DataSnap, nous devons maintenant aussi déployer les pilotes de base de données. Les pilotes et fichiers dépendent de la base de données sélectionnée. En utilisant DBX4, pensez à vérifier le composant TSQLConnection et les fichiers dbxconnections.ini et dbxdrivers.ini dans C:\Documents and Settings\All Users\ Documents\RAD Studio\dbExpress\7.0 (Windows XP) ou dans C:\Users\Public\Documents\RADStudio\dbExpress\7.0 (Windows Vista et Windows 7).
Le fichier dbxdrivers.ini indique pour le pilote donné les composants DriverPackageLoader et MetaDataPackageLoader (pointant généralement vers le même package). Pour BlackfishSQL, cela signifie que DBXClientDriver140.bpl doit être déployé ainsi que Blackfish. Pour plus d'informations sur le déploiement de BlackfishSQL, reportez-vous fichier deploy_en.htm dans le répertoire RADStudio\7.0.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

3.4. REUTILISATION DES MODULES DE DONNEES DISTANTS EXISTANTS

S'il existe des classes TRemoteDataModule, vous pouvez toujours les utiliser conjointement avec le nouveau DataSnap. Mais vous devez supprimer quelques fonctionnalités du serveur (spécialement les éléments COM).
En premier lieu, si vous souhaitez migrer une application Serveur DataSnap existante et pas seulement le module de données distant, vous devez désinscrire le serveur DataSnap en lançant l'exécutable en ligne de commande avec l'option de ligne de commande /unregister. Si vous ne faites pas cela depuis le début, vous ne serez pas en mesure de désinscrire le module de données distant du registre (sauf si vous restaurez ultérieurement une sauvegarde du projet).
Dans l'unité pour le module de données distant, nous devons supprimer le code de la section d'initialisation. Pour conserver une unité compatible entre Delphi 2007 (ou antérieur) et 2009 (ou postérieur), vous pouvez placer ce code dans {$IFDEF} comme suit :

```
{$IF CompilerVersion >= 20}
initialization
    TComponentFactory.Create(ComServer, TRemoteDataModule2010,
        Class RemoteDataModule2010, ciMultiInstance, tmApartment);
{$IFEND}
end.
```

Nous devons également supprimer la routine UpdateRegistry du projet ou la placer également dans {$IFDEF}.

```
{$IF CompilerVersion >= 20}
class procedure UpdateRegistry(Register: Boolean;
    const ClassID, ProgID: string); override;
{$IFEND}
```

Le changement le plus important pour réaliser l'opération consiste à supprimer la bibliothèque de type (ou fichiers .ridl) et l'unité d'importation de bibliothèque de type. Ils ne peuvent pas demeurer dans {IFDEF}s, par conséquent, si vous devez conserver une version Delphi 2007 ou antérieure (COM) et Delphi 2009 ou ultérieure (sans COM) du serveur DataSnap vous devrez faire maintenant une copie du projet. Nous devons utiliser un composant TDSServerClass de l'application serveur DataSnap et retourner la classe TRemoteDataModule, comme nous avons fait précédemment.
Finalement, nous devons nous assurer que toutes les méthodes personnalisées qui ont été ajoutées à TRemoteDataModule sont déplacées de la section protégée (DataSnap COM par défaut) à la section publique (l'information méthode est générée dans l'architecture DataSnap sans COM).

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

4. FILTRES DATASNAP – LES DONNÉES COMME VOUS VOULEZ

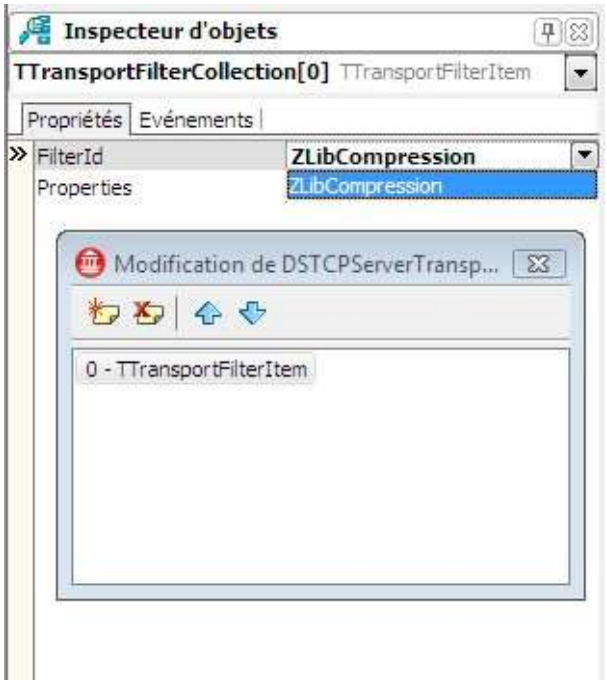
Dans cette section, nous aborderons le fonctionnement des filtres et la façon d'utiliser les filtres existants (tels que la compression) ou de construire de nouveaux filtres DataSnap. Les filtres DataSnap sont des DLL spécifiques qui interceptent le flux d'octets de communication et peuvent fonctionner dans une chaîne de filtres complète.
Nous pouvons par exemple combiner compression et cryptage, journalisation et compression, etc.
Nous devons indiquer quel(s) filtre(s) doivent être utilisés par le serveur et le client DataSnap à deux endroits. Pour le serveur, nous devons indiquer une liste dans la propriété Filters du composant TDSTCPServerTransport et pour les clients, nous devons indiquer une liste similaire de filtres dans les clauses d'utilisation du projet client DataSnap. Cela suffit pour le client dans la mesure où chaque filtre DataSnap doit automatiquement s'enregistrer pour utilisation.
Lors du gestionnaire d'événements OnConnect, nous pouvons examiner les filtres enregistrés utilisés pour la connexion (par exemple en utilisant une fonction spécifique LogInfo pour écrire ces informations dans un fichier journal) :

```
procedure TServerContainer1.DSServer1Connect(
     DSConnectEventObject: TDSConnectEventObject);
var
     i: Integer;

begin
     LogInfo('Connect ' + DSConnectEventObject.ChannelInfo.Info);
     for i:=0 to DSConnectEventObject.Transport.Filters.Count-1 do
     LogInfo(' Filter: ' +
     DSConnectEventObject.Transport.Filters.GetFilter(i).Id);

end;
```

4.1. FILTRE ZLIBCOMPRESSION

Examinons par exemple un filtre DataSnap existant, livré avec Delphi 2010 qui peut être utilisé pour comprimer les flux de données entre le serveur DataSnap et l'unité DbxCompressionFilter.
Le composant TDSTCPServerTransport (pour TCP/IP) et le composant DSHTTPService (pour HTTP) ont une propriété Filters contenant TTransportFiltersCollection. Nous pouvons cliquer sur l'ellipse de la propriété Filters pour modifier la collection de filtres. Dans ce dialogue, nous pouvons ajouter un nouvel élément TTransportFilterItem et utiliser l'inspecteur d'objets pour paramétrer FilterId et quelques propriétés optionnelles. Delphi 2010 est livré avec le filtre ZLibCompression qui peut être spécifié ici comme FilterId.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



Remarquons qu'en plus de la propriété Filters du composant TDSTCPServerTransport côté serveur, nous devons également spécifier que nous souhaitons également utiliser ce filtre côté client (pour compresser les requêtes sortantes et décompresser les réponses entrantes). Pour cela, nous devons seulement ajouter l'unité DbxCompressionFilter à la clause d'utilisation de ClientForm. Ceci enregistrera automatiquement TTransportCompressionFilter pour s'assurer qu'il sera utilisé pour communiquer avec le serveur.
Si vous n'ajoutez pas l'unité DbxCompressionFilter à la clause d'utilisation, l'exécution du client générera une exception avec le message suivant :

4.2. FILTRE DE JOURNALISATION (LOG)

Delphi 2010 DataSnap est ouvert pour nous permettre de définir nos propres filtres de transport. Nous pouvons **réaliser** cette opération en dérivant une nouvelle classe du type TTransportFilter. Dans cette nouvelle classe, nous pouvons éviter les méthodes de base et les implémenter. Nous pouvons par exemple créer une classe TLogFilter comme suit :

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

```
unit LogFilter;
interface
uses
SysUtils, DBXPlatform, DBXTransport;
type
TLogFilter = class(TTransportFilter)
private
protected
function GetParameters: TDBXStringArray; override;
function GetUserParameters: TDBXStringArray; override;
public
function GetParameterValue(const ParamName: UnicodeString): UnicodeString; override;
function SetParameterValue(const ParamName: UnicodeString;
const ParamValue: UnicodeString): Boolean; override;
constructor Create; override;
destructor Destroy; override;
function ProcessInput(const Data: TBytes): TBytes; override;
function ProcessOutput(const Data: TBytes): TBytes; override;
function Id: UnicodeString; override;
end;
const
LogFilterName = 'Log';
```

L'implémentation de cette classe peut demeurer vierge pour l'essentiel. Dans la mesure où le seul objectif du filtre est de journaliser les données émises lors des méthodes ProcessInput et ProcessOutput, nous pouvons laisser la plupart des autres méthodes vides. L'implémentation de méthodes non-vides s'effectue comme suit :

```
function TLogFilter.SetParameterValue(const ParamName, ParamValue: UnicodeString): Boolea
begin
Result := True;
end;
constructor TLogFilter.Create;
begin
inherited Create;
end;
destructor TLogFilter.Destroy;
begin
inherited Destroy;
end;
function TLogFilter.ProcessInput(const Data: TBytes): TBytes;
begin
Result := Data; // journalisation données entrantes
end;
```

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

```
function TLogFilter.ProcessOutput(const Data: TBytes): TBytes;
begin
Result := Data; // journalisation données sortantes
end;
function TLogFilter.Id: UnicodeString;
begin
Result := LogFilterName;
end;
```

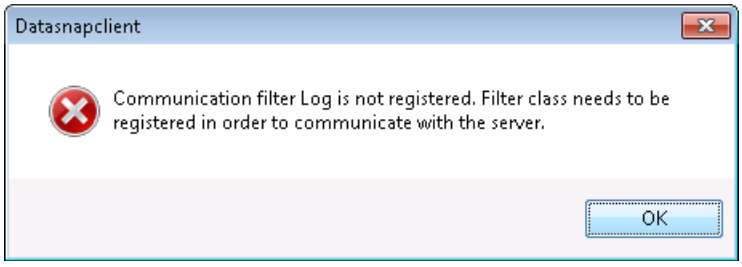
Finalement, une partie importante de l'implémentation du filtre de transport DataSnap est la partie enregistrement de la section d'initialisation et de finalisation. Ceci permet de s'assurer que le client DataSnap pourra « trouver » le filtre de transport et l'utiliser quand cela est nécessaire.

```
initialization
TTransportFilterFactory.RegisterFilter(LogFilterName, TLogFilter);
finalization
TTransportFilterFactory.UnregisterFilter(LogFilterName);
end.
```

Comme nous l'avons vu, pour utiliser le filtre transport dans le serveur DataSnap, nous devons l'ajouter à la liste des filtres de TDSTCPServerTransport ou du composant TDSHTTPService.
En phase de conception, le filtre ZLibCompression est déjà connu mais pas les nouveaux filtres (sauf si nous les ajoutons à un package pour les installer). Heureusement, nous pouvons également ajouter les filtres de transport en exécution en ajoutant l'unité filtre à la clause d'utilisation de ServerContainerUnitDemo puis en ajoutant le filtre manuellement (par nom) à la liste des filtres, par exemple dans :

```
procedure TServerContainer1.DataModuleCreate(Sender: TObject);
begin
DSTCPServerTransport1.Filters.AddFilter(LogFilterName);
DSHTTPService1.Filters.AddFilter(LogFilterName);
DSHTTPService1.Active := True;
end;
```

Ceci permet de s'assurer que le serveur utilise LogFilter et que le client l'utilisera automatiquement après l'ajout de l'unité LogFilter à la clause d'utilisation du client. Sinon, le message d'erreur suivant s'affichera :



Remarquons que chaque application – serveur et clients DataSnap – obtient son propre fichier de log et bien que le même filtre de journalisation soit utilisé, nous

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

n'avons pas à ajouter d'information telle que ParamStr(0) pour identifier quelle cible produit effectivement le message de log.

4.3. Filtre de cryptage

Après l'exemple simple de la section 4.2, il est clair que le développement de filtres personnalisés plus complexes n'est pas réellement compliqué. En fait, il existe déjà certains filtres tiers (voir notamment http://www.danieleteti.it/?p=168 contenant 9 filtres additionnels pour DataSnap 2010 classés en trois groupes : « Hash » prenant en charge MD5, MD4, SHA1 et SHA512 ; « Cipher » prenant en charge Blowfish, Rijndael, 3TDES et 3DES et « Compress » prenant en charge LZO. L'ensemble du code source est disponible.

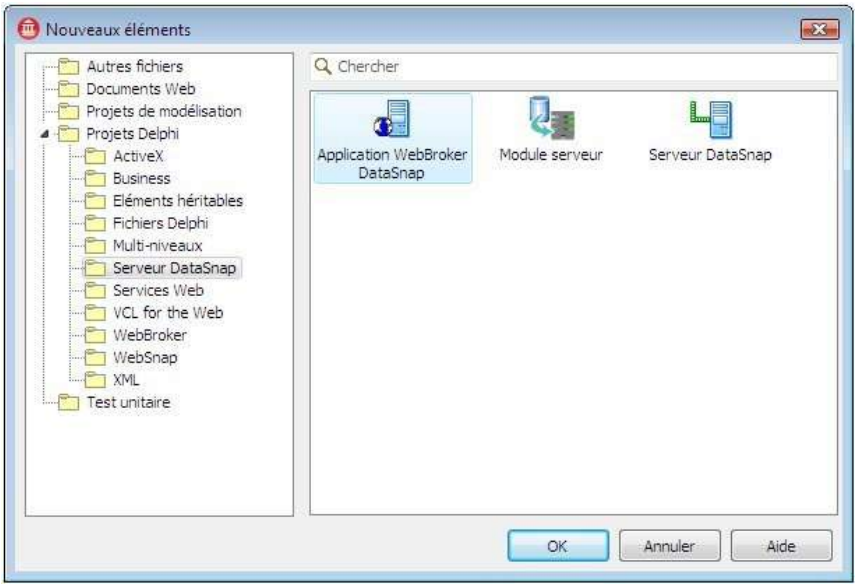
5. Cibles Web DataSnap – comme vous voulez (suite)

Au-delà des cibles Windows, un assistant permet de produire des cibles ISAPI, CGI ou Exécutable débogueur d'application Web. Nous aborderons d'abord les avantages de ces cibles et démontrerons comment produire un groupe de projet unique avec trois cibles différentes partageant les mêmes unités personnalisées ; il en résulte un seul groupe de projets avec trois projets produisant une cible différente pour le même objet serveur DataSnap.

Bien que les applications serveur DataSnap que nous avons conçues fonctionnent correctement, à certains moments il est impossible de déployer ces applications serveur. Par exemple, lorsqu'il est impossible ou interdit d'ouvrir les ports requis du pare-feu pour permettre aux clients de se connecter au serveur. Cependant, dans la plupart des cas, un site Web est hébergé sur un serveur Web et le port 80 est généralement ouvert (pour le serveur Web).

Si nous supposons que Microsoft Internet Information Services (IIS) est utilisé comme serveur Web, nous pouvons utiliser le nouvel assistant Application DataSnap WebBroker pour produire un projet qui sera déployé sur IIS.

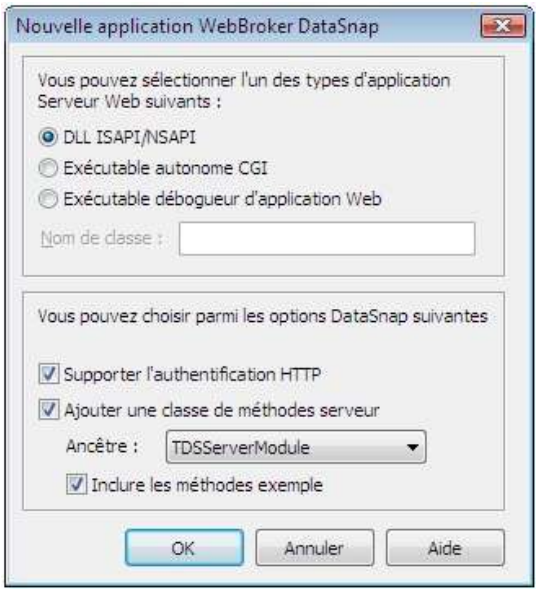
Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



Cet assistant propose trois choix dont l'un n'est pas véritablement une application WebBroker mais essentiellement un client de débogage d'application Web à n'utiliser qu'à cette fin. Ce client est très puissant puisqu'il permet d'utiliser le Débogueur d'application Web (dans le menu Outils de l'EDI Delphi) comme application hôte tout en déboguant l'application Client DataSnap avec le débogueur d'application Web.

Le débogage des applications CGI ou ISAPI/NSAPI est plus délicat. Par conséquent le Débogueur d'application Web est un choix pertinent pour les applications encore en développement.

Les cibles restantes (DLL ISAPI/NSAPI/Exécutable autonome CGI) peuvent être sélectionnées pour de véritables projets serveur DataSnap.



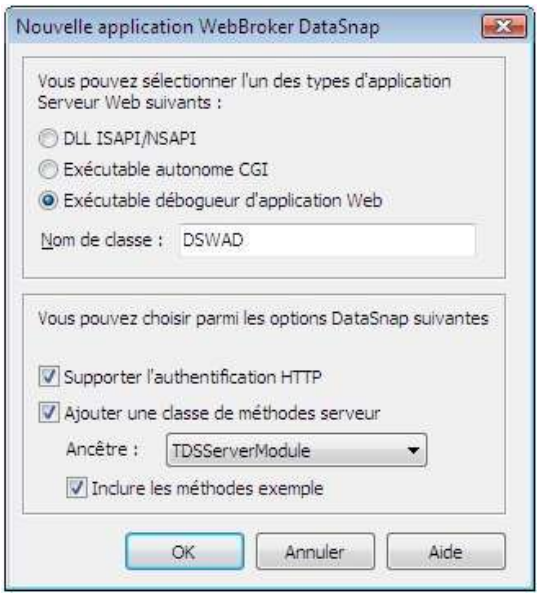
Remarquons que le choix de Exécutable autonome CGI n'est pas une excellente idée car cet exécutable sera chargé/déchargé à chaque requête entrante. En ajoutant le temps de connexion à une base de données pour effectuer quelques opérations, on comprend bien les problèmes de performance qui peuvent se poser. L'utilisation d'une cible ISAPI résultera dans une DLL qui n'est chargée qu'une fois et demeure en mémoire afin que les requêtes suivantes (d'autres

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

utilisateurs) ne souffrent pas d'une pénalité de performance additionnelle. L'inconvénient majeur d'une DLL ISAPI est qu'il n'est pas simple de remplacer (si vous n'avez qu'un accès FTP au serveur Web) mais il existe suffisamment de gestionnaires ISAPI pour solutionner ce problème à votre place (pour plus de détails contactez votre fournisseur d'hébergement Web).
L'autre inconvénient d'une cible ISAPI DLL réside dans les difficultés de débogage, exigeant de charger IIS comme application hôte – ce qui ne fonctionne pas toujours comme prévu. Cependant, ce problème particulier est résolu par la présence d'un exécutable Débogueur d'application Web – il suffit de s'assurer d'utiliser deux projets utilisant les mêmes code et méthodes personnalisées DataSnap. Il s'agit d'un bon point de départ pour une première démonstration, ajoutant des techniques réelles pour obtenir un squelette opérationnel.

5.1. DÉBOGUEUR D'APPLICATION WEB

Utilisons tout d'abord l'Assistant Nouvelle application WebBroker DataSnap pour créer une nouvelle application Exécutable débogueur d'application Web. Indiquons par exemple DSWAD pour le nom de classe et cochons l'option Supporter l'authentification HTTP.



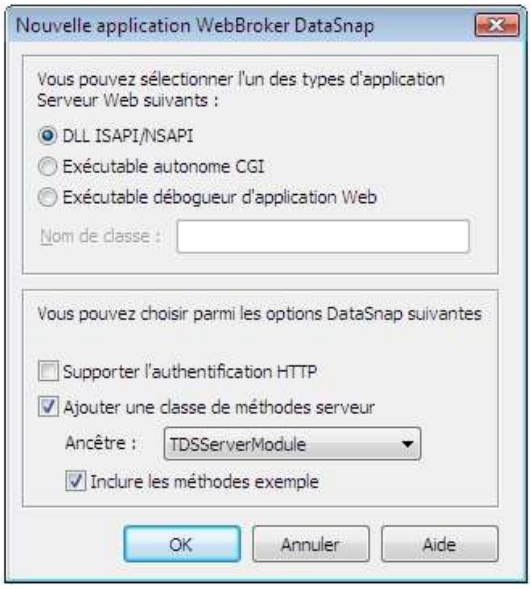
Après avoir cliqué sur OK, un nouveau projet est créé avec trois unités. En l'absence de fichiers Project1 et Unit1 dans le répertoire de projets par défaut, le projet sera dénommé Project1 et les unités Unit1, ServerMethodsUnit1 et Unit2. La première unité doit être un formulaire vide – unique pour l'Exécutable débogueur d'application Web et non requis pour les autres cibles Web. Enregistrez cette unité dans WADForm.pas. La deuxième unité est dénommée ServerMethodsUnit1.pas et contient notre module serveur dérivé de TDSServerModule comme indiqué dans la boîte de dialogue. Nous reviendrons à cette unité dans un moment ; nous pouvons l'enregistrer pour l'instant sous le nom ServerMethodsUnit1.pas. La troisième unité dénommée Unit2 est un module Web à trois composants déjà présents (trois, si vous n'avez pas coché l'option

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Supporter l'authentification HTTP). Cette unité doit recevoir les requêtes entrantes et les distribuer aux modules du serveur DataSnap du projet. Enregistrez cette unité dans DSWebMod.pas.
Enfin, enregistrez le projet sous DSWADServer.dproj pour indiquer qu'il s'agit du serveur DataSnap Débogueur d'application Web.

5.2. CIBLE ISAPI

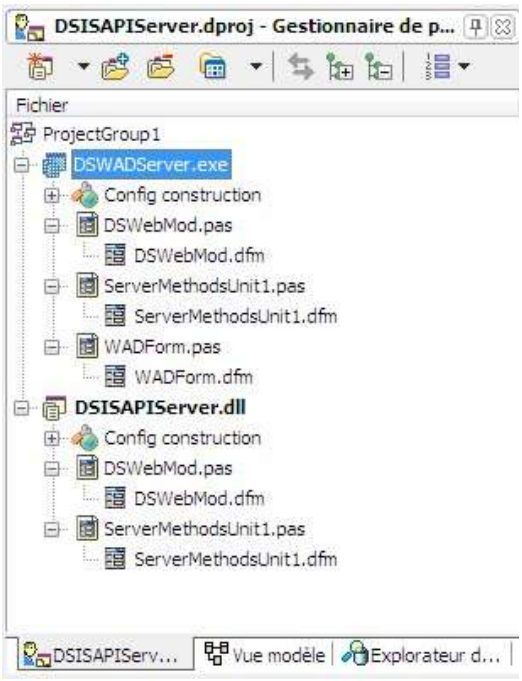
Avant de poursuivre la modification et la personnalisation des unités ServerMethodsUnit1.pas et DSWebMod.pas, nous devons tout d'abord ajouter un nouveau projet au groupe – cette fois une application DLL ISAPI/NSAPI. Cliquez avec le bouton droit sur ProjectGroup et sélectionnez Ajouter un nouveau projet pour obtenir le référentiel objet et démarrer un nouveau projet. Dans la catégorie Serveur DataSnap, utilisez à nouveau l'assistant Nouvelle application WebBroker DataSnap pour créer une nouvelle application DLL ISAPI/NSAPI. Cette fois, vous n'avez à modifier aucune option dans la partie inférieure de la boîte de dialogue puisque nous allons réutiliser les unités existantes du projet DSWADServer.



Cliquez sur OK pour générer un nouveau projet (qui sera ajouté au groupe), nommé Project1, avec les unités ServerMethodUnit2.pas et Unit1.pas ajoutées au nouveau projet.
Maintenant, au lieu d'utiliser les nouvelles unités ServerMethodUnit2.pas et Unit1.pas, nous devons utiliser ServerMethodUnit1.pas et DSWebMod.pas qui font déjà parti du projet DSWADServer. Cliquez avec le bouton droit sur le nœud ServerMethodUnit2.pas sous le nœud Project1.dll et sélectionnez Retirer du Projet. Cliquez sur OK dans la boîte de dialogue de confirmation (remarquez que si vous ne les avez pas encore sauvegardées, il n'est naturellement pas utile de supprimer les fichiers ServerMethodUnit2.pas et .dfm du disque). Faites de même avec Unit1.pas, afin que Project1.dll ne contienne plus d'unités. Ensuite, cliquez avec le bouton droit sur Project1.dll et ajoutez les unités DSWebMod.pas et ServerMethodUnit1.pas au projet. Finalement, renommez Project1 en DSISAPIServer.dproj pour compléter le groupe de projet.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Vous devriez avoir maintenant un groupe avec deux projets partageant les unités DSWebMod et DSSererMethodUnit1.pas comme présenté dans la copie d'écran suivante :



Cette configuration permet de construire deux projets utilisant DSWADServer comme cible pour les tests et le débogage et DSISAPIServer pour le déploiement effectif du serveur DataSnap sur IIS.
Avant de poursuivre en ajoutant des méthodes Web à ServerMethodsUnit1, nous devons tout d'abord corriger le projet ISAPI/NSAPI en supprimant du fichier le code créant une instance automatique de TDSServerModule. En effet TDSServerModule étant un module de données, nous obtiendrons un message d'erreur si nous tentons d'exécuter la DLL ISAPI car il ne peut exister qu'un module Web dans l'application.
Ouvrez le code source du projet DSISAPIServer.dpr et modifiez le bloc principal comme suit :

```
begin
CoInitFlags := COINIT MULTITHREADED;
Application.Initialize;
Application.CreateForm(TWebModule2, WebModule2);
// Application.CreateForm(TServerMethods1, ServerMethods1);
Application.Run;
end.
```

Ceci permettra d'éviter le message d'erreur indiquant qu'un seul module de données est autorisé. Remarquons que vous pouvez ne pas voir ce message d'erreur lors de l'appel du composant ISAPI DLL (déployé) mais une erreur serveur ou de temporisation – c'est pourquoi il est important de se souvenir de ce problème.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

5.3. METHODES SERVEUR, DEPLOIEMENT ET CLIENTS

Lors de l'ajout de fonctionnalités, nous devons travailler uniquement sur l'unité ServerMethodUnit1.pas qui est partagée par les deux cibles. Par défaut, une méthode exemple est déjà incluse mais comme pour les versions Windows du serveur DataSnap, nous pouvons ajouter deux autres méthodes (voir à la section 2.1.4 pour les composants et le code source nécessaires dans l'unité Méthodes serveur).

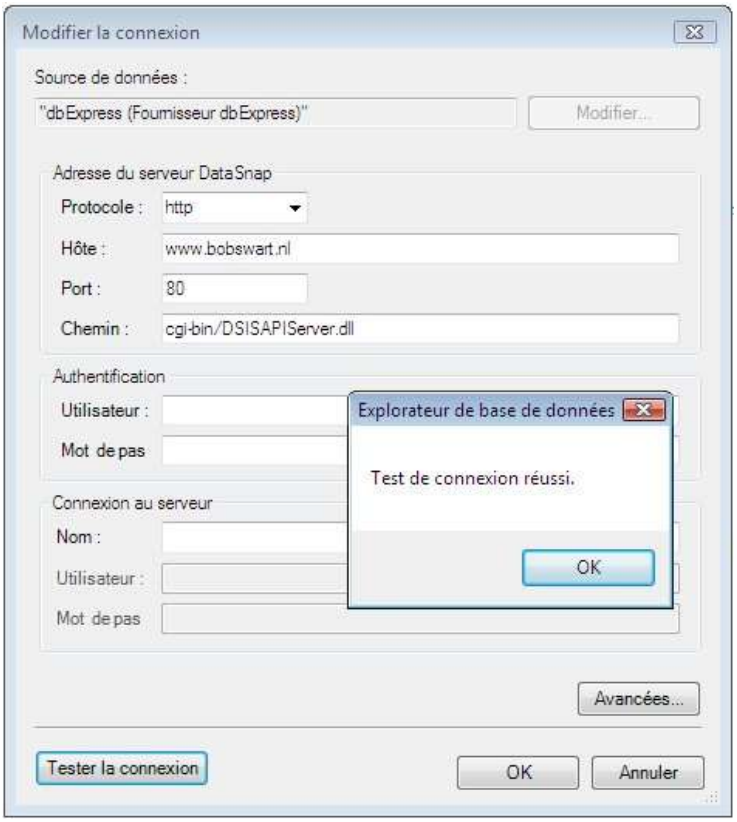
Lorsque les méthodes serveur sont implémentées, nous pouvons déployer ISAPI DLL sur un serveur Web comme Microsoft IIS. Ceci est expliqué en détail dans un article de Jim Tierney sur http://blogs.embarcadero.com/jimtierney/2009/08/20/31502 nous n'y reviendrons donc pas ici.

Si vous ne disposez pas de serveur Web de déploiement, vous pouvez manipuler le serveur DataSnap ISAPI déployé sur mon serveur. Remarquons que TDataSetProvider n'est pas exposé et que les données ne sont pas retournées dans la méthode GetEmployees mais que les méthodes ServerTime et EchoString fonctionnent parfaitement – ce qui doit suffire pour écrire un client DataSnap de test sur ce serveur.

Avant de connecter le serveur ISAPI DataSnap à l'intérieur d'une application client, il est pertinent d'utiliser l'Explorateur de donnés pour valider la connexion au serveur ISAPI DataSnap. L'explorateur de donnés dispose d'une nouvelle catégorie dénommée DATASNAP, si vous l'ouvrez, une première connexion dénommée DATASNAPCONNECTION est disponible que vous pouvez modifier (en cliquant avec le bouton droit et en sélectionnant Modifier la connexion).

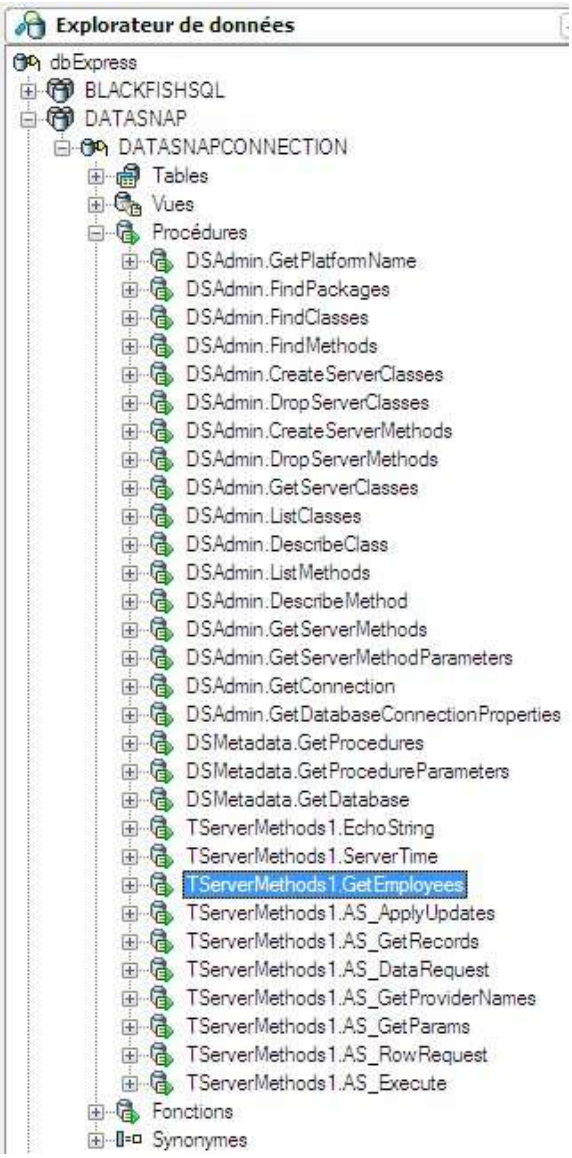
Dans cette boîte de dialogue, nous pouvons indiquer le protocole, l'hôte (vous pouvez utiliser www.bobswart.nl si vous n'avez pas votre propre serveur Web), le port et le chemin URL vers l'application serveur ISAPI DataSnap (cgi-bin/DSISAPIServer.dll). Cliquez sur Tester la connexion pour vous assurer que tout fonctionne.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



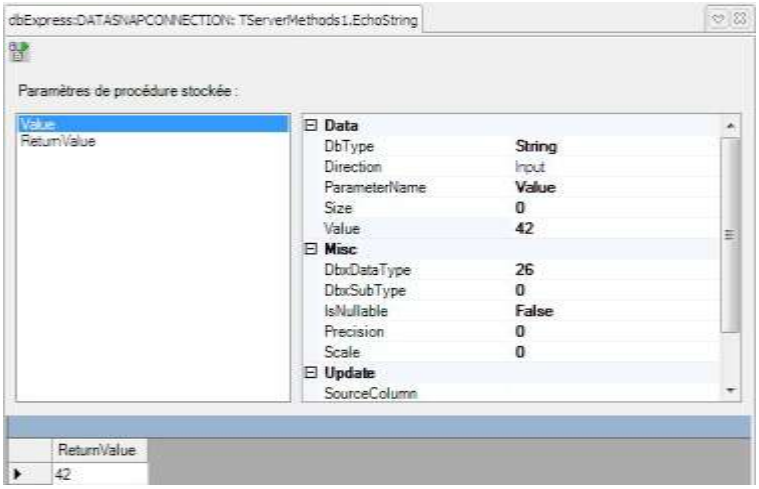
Cliquez maintenant sur OK pour refermer la boîte de dialogue. Dans l'explorateur de données, vous pouvez désormais développer le nœud DATASNAPCONNECTION pour afficher les tables, vues, procédures, fonctions et synonymes. Vous constaterez que les procédures incluent tous les DSAdmin, DSMetaData, TServerMethods1.AS_xxx ainsi que nos trois méthodes serveur personnalisées : EchoString, ServerTime et GetEmployees.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



Sans qu'il soit nécessaire d'écrire une application client DataSnap, nous pouvons désormais tester certaines méthodes. Par exemple EchoString (pour s'assurer du retour de ce que nous envoyons).
Si nous cliquons avec le bouton droit sur la procédure TServerMethods1.EchoString, nous pouvons choisir Afficher les paramètres pour afficher une nouvelle fenêtre dans l'EDI où nous pourrons saisir la valeur du paramètre de Value (par exemple 42). Ensuite, en cliquant avec le bouton droit sur cette nouvelle fenêtre, nous pouvons sélectionner Exécuter pour exécuter la méthode serveur distante. Le résultat est affiché ci-dessous :

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



Ceci démontre que nous pouvons appeler les méthodes serveur personnalisées depuis le serveur DataSnap. Pour connecter l'application client DataSnap au serveur, il suffit de modifier les propriétés de TSQLConnection. Précédemment, nous nous sommes connectés à la version Windows du serveur DataSnap ; nous devons maintenant modifier ces paramètres pour se connecter à la version Web.

Rappelez vous que si vous désirez réellement utiliser DSISAPIServer.dll de mon serveur Web que j'ai désactivé TDataSetProvider et ne retourne aucune valeur dans les méthodes GetEmployees mais vous pouvez appeler les méthodes ServerTime et EchoString.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

6. REST ET JSON...
COMME VOUS VOULEZ

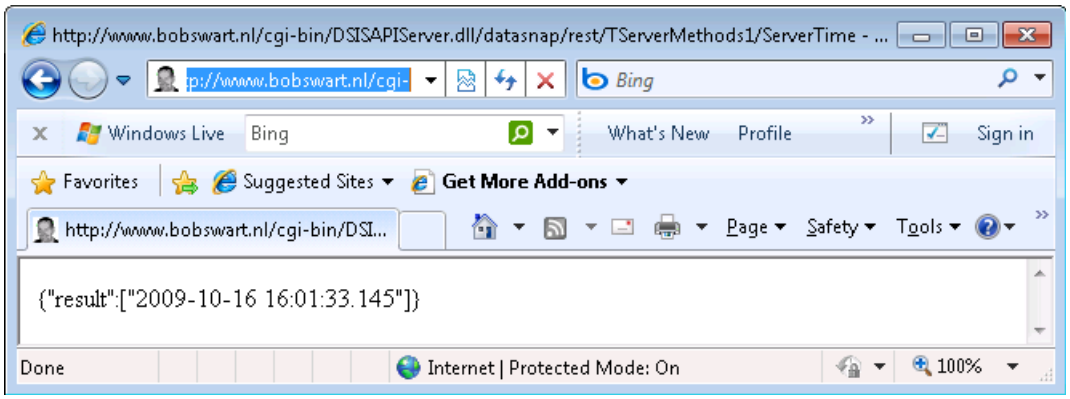
DataSnap 2010 prend en charge REST et JSON. DataSnap 2010 prend en charge REST pour les requêtes DataSnap HTTP. Par exemple, si l'URL du serveur DataSnap est http://www.bobswart.nl/cgi-bin/DSISAPIServer.dll, nous pouvons y ajouter /datasnap/rest, suivi du nom de la classe Server Method, de la méthode et des arguments.
La syntaxe générique se présente comme suit :

http://server/datasnap/rest/<class>/<method>/<parameters>

Pour la méthode ServerTime du module TServerMethod1 de DSISAPIServer.dll sur mon serveur, l'URL se présente comme suit :

http://www.bobswart.nl/cgi-bin/DSISAPIServer.dll/datasnap/rest/TServerMethods1/ServerTime

L'appel de cette URL REST produit un résultat JSON, par exemple dans le navigateur :



Le résultat affiché dans le navigateur est un construit JSON :

```
{"result":["2009-10-16 16:01:33.145"]}
```

Marco Cantù fournit d'autres détails sur REST dans son livre blanc consacré à Delphi 2010 et aux clients REST.

6.1. RAPPELS (CALLBACKS)

En plus d'être le résultat d'appels REST aux serveurs DataSnap, JSON est également utilisé lors de l'implémentation de méthodes de rappel. DataSnap 2010 prend en charge les fonctions de rappel côté client, exécutées dans le contexte d'une méthode serveur. Cela signifie que lors de l'exécution d'une méthode serveur (appelée par le client DataSnap), le serveur peut appeler une fonction de rappel passée en tant qu'argument par le client à la méthode serveur.
Par exemple, modifions la méthode EchoString afin de lui ajouter une fonction de rappel.
La définition de la méthode EchoString doit être modifiée comme suit :

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

```
function EchoString(Value: string; callback: TDBXcallback): string;
```

Le type de TDBXcallback est défini dans l'unité DBXJSON. Avant de pouvoir implémenter la nouvelle méthode EchoString, nous devons tout d'abord examiner comment la méthode de rappel peut être définie côté client (après tout, il s'agit d'une méthode client pouvant être appelée par le serveur). Côté client, nous devons déclarer une nouvelle classe, dérivée de TDBXCallback et éviter la méthode Execute.

```
type
TCallbackClient = class(TDBXCallback)
public
function Execute(const Arg: TJSONValue): TJSONValue; override;
end;
```

Dans la méthode Execute, nous obtenons l'argument Arg de type TJSONValue, que nous pouvons cloner pour manipuler le véritable contenu. La méthode Execute peut également retourner lui-même TJSONValue.

```
function TCallbackClient.Execute(const Arg: TJSONValue): TJSONValue;
var
Data: TJSONValue;
begin
Data := TJSONValue(Arg.Clone);
ShowMessage('Callback: ' + TJSONObject(Data).Get(0).JSonValue.value);
Result := Data
end;
```

Dans cet exemple, la méthode de rappel indique la valeur passée à la méthode EchoString, avant que la méthode n'effectue un véritable retour (c'est-à-dire pendant qu'elle est encore en cours d'exécution). L'implémentation de la nouvelle méthode EchoString côté serveur doit désormais disposer la valeur de chaîne dans un objet TJSONObject et la passer à la méthode callback.Execute comme suit :

```
function TServerMethods2.EchoString(Value: string; callback: TDBXcallback): string;
var
msg: TJSONObject;
pair: TJSONPair;
begin
Result := Value;
msg := TJSONObject.Create;
pair := TJSONPair.Create('ECHO', Value);
pair.Owned := True;
msg.AddPair(pair);
callback.Execute(msg);
end;
```

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Remarquons que la fonction de rappel est exécutée (côté client) – et retourne – avant que la méthode EchoString n'ait terminé côté serveur.
Enfin, l'appel à la méthode EchoString côté client doit également être modifié puisque nous devons maintenant passer une classe de rappel – une instance du nouveau TCallbackClient – comme second argument.

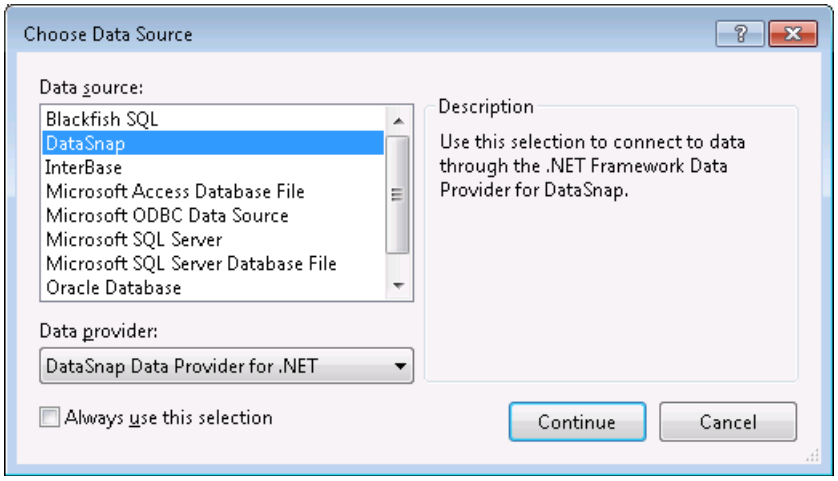
```
var
MyCallback: TCallbackClient;
begin
MyCallback := TCallbackClient.Create;
try
Server.EchoString(Edit1.text, MyCallback);
finally
MyCallback.Free;
end;
end;
```

Ce simple exemple démontre comment utiliser des méthodes de rappel côté client dans DataSnap 2010.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

7. DataSnap et .NET – ou vous voulez (suite)

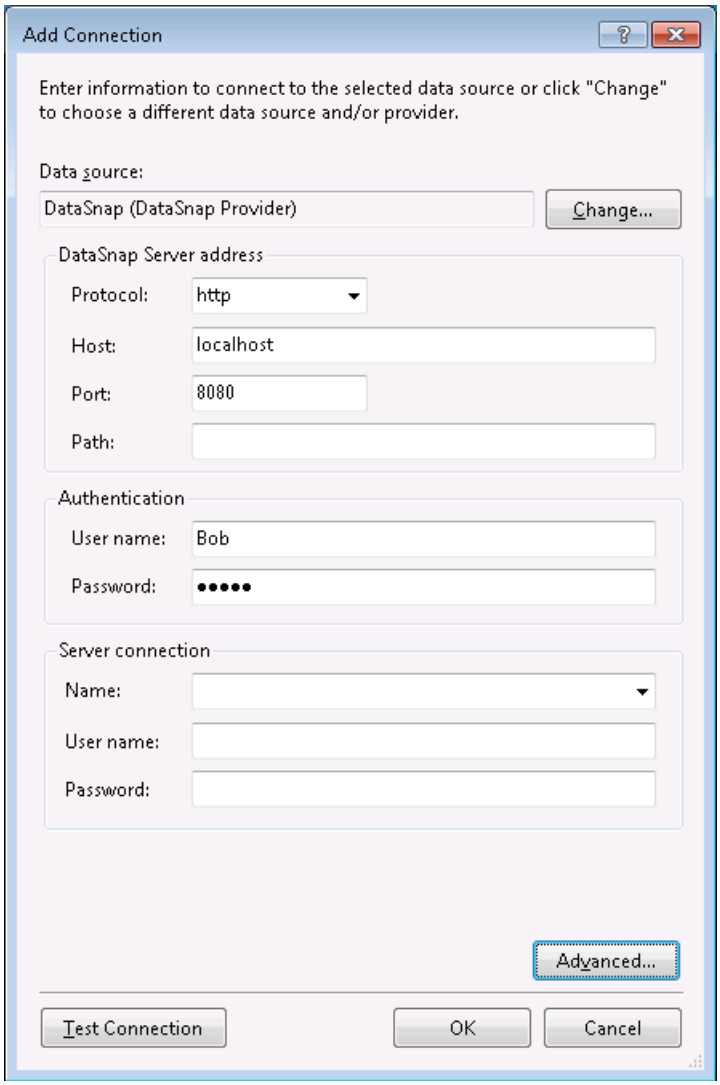
Delphi Prism 2010 est utilisé pour construire un client DataSnap .NET pour les serveurs Win32 conçus jusqu'à présent. Pour construire le client Delphi Prism 2010 DataSnap, assurez-vous qu'un serveur DataSnap est opérationnel pour que nous puissions nous y connecter dès la phase de conception.
Démarrez Delphi Prism 2010, et choisissez View | Server Explorer pour afficher l'explorateur serveur Delphi Prism. Nous devons tout d'abord réaliser une connexion pour vérifier que nous pouvons vraiment travailler avec le serveur DataSnap.
L'explorateur serveur affiche une arborescence avec un nœud racine dénommé Data Connections. Cliquez avec le bouton droit sur Data Connections et sélectionnez Add Connection. Dans la boîte de dialogue qui suit, sélectionnez DataSnap dans la liste des sources de données (remarquez que vous devez cliquer sur Change si une source de données est déjà présélectionnée).



Vous pouvez décocher la boîte « Always use this selection » sauf bien sûr si vous ne voulez construire que des connexions de données DataSnap.
Cliquez sur Continue pour obtenir la page suivante du dialogue. Vous pouvez alors indiquer les détails de connexion au serveur DataSnap. Dans la liste de choix Protocol, nous pouvons choisir tcp/ip ou http. Ensuite, vous devez indiquer l'hôte (c'est-à-dire le nom de la machine sur laquelle le serveur DataSnap est exécuté – éventuellement localhost si vous effectuez les tests sur la même machine locale). Ensuite, vous devez indiquer le numéro de port. Par défaut, il s'agira du port 80 pour HTTP et du port 211 pour TCP/IP, mais vous savez maintenant que les deux valeurs peuvent (ou doivent) être différentes – assurez-vous au moins d'indiquer la même valeur ici que celle indiquée dans le/les composants de transport de l'unité ServerContainerUnitDemo.
La propriété suivante contient le chemin d'accès. Il n'est important que si vous devez vous connecter à un serveur DataSnap basé sur Web Broker (où vous devez indiquer le chemin URL pour accéder au serveur Web DataSnap – c'est-à-dire, la partie après the http://..../ domaine.

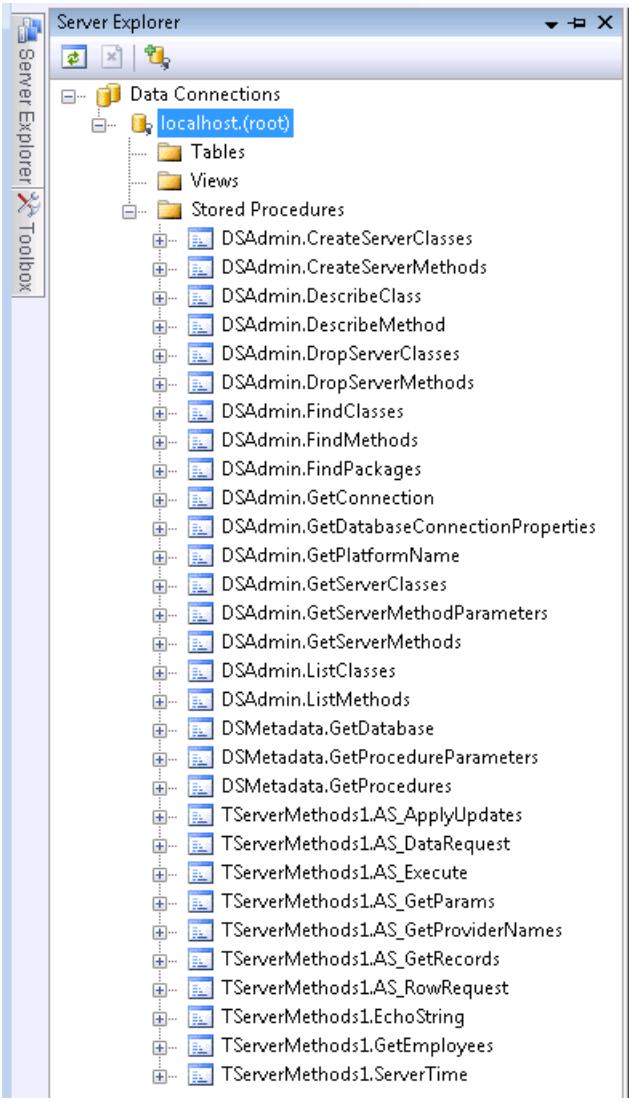
Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

Finalement, n'oubliez pas d'indiquer le nom d'utilisateur et le mot de passe d'authentification si le serveur DataSnap utilise l'authentification HTTP.

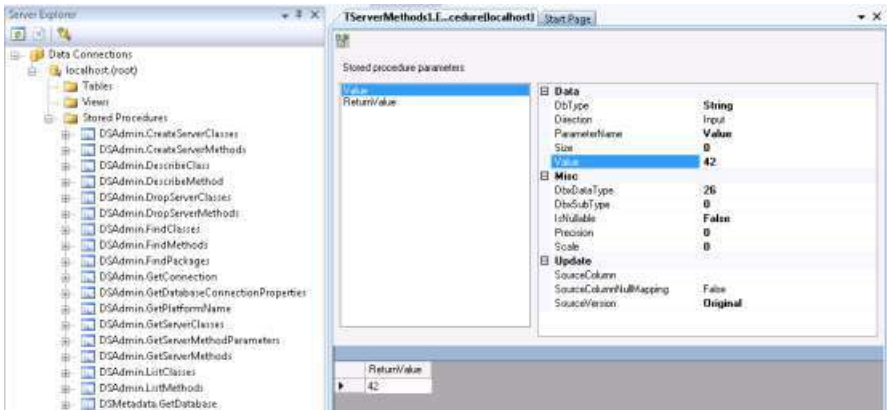


Cliquez sur le bouton de test de connexion pour vérifier que la connexion peut être réalisée vers le serveur DataSnap indiqué. Ceci vous permettra d'afficher le message « Test connection succeeded » si tout a été indiqué correctement. Lorsque vous cliquez sur OK, une nouvelle entrée pour la connexion DataSnap sera ajoutée à l'arbre des connexions aux données (dans ce cas pour un nœud localhost). Si vous devez développer le nouveau nœud, vous trouverez des sous-nœuds pour Tables, Views et Stored Procedures ; les deux premiers sont vides mais Stored Procedures doit contenir toutes les méthodes serveur exposées du serveur DataSnap — y compris nos méthodes personnalisées EchoString, GetEmployees et ServerTime.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

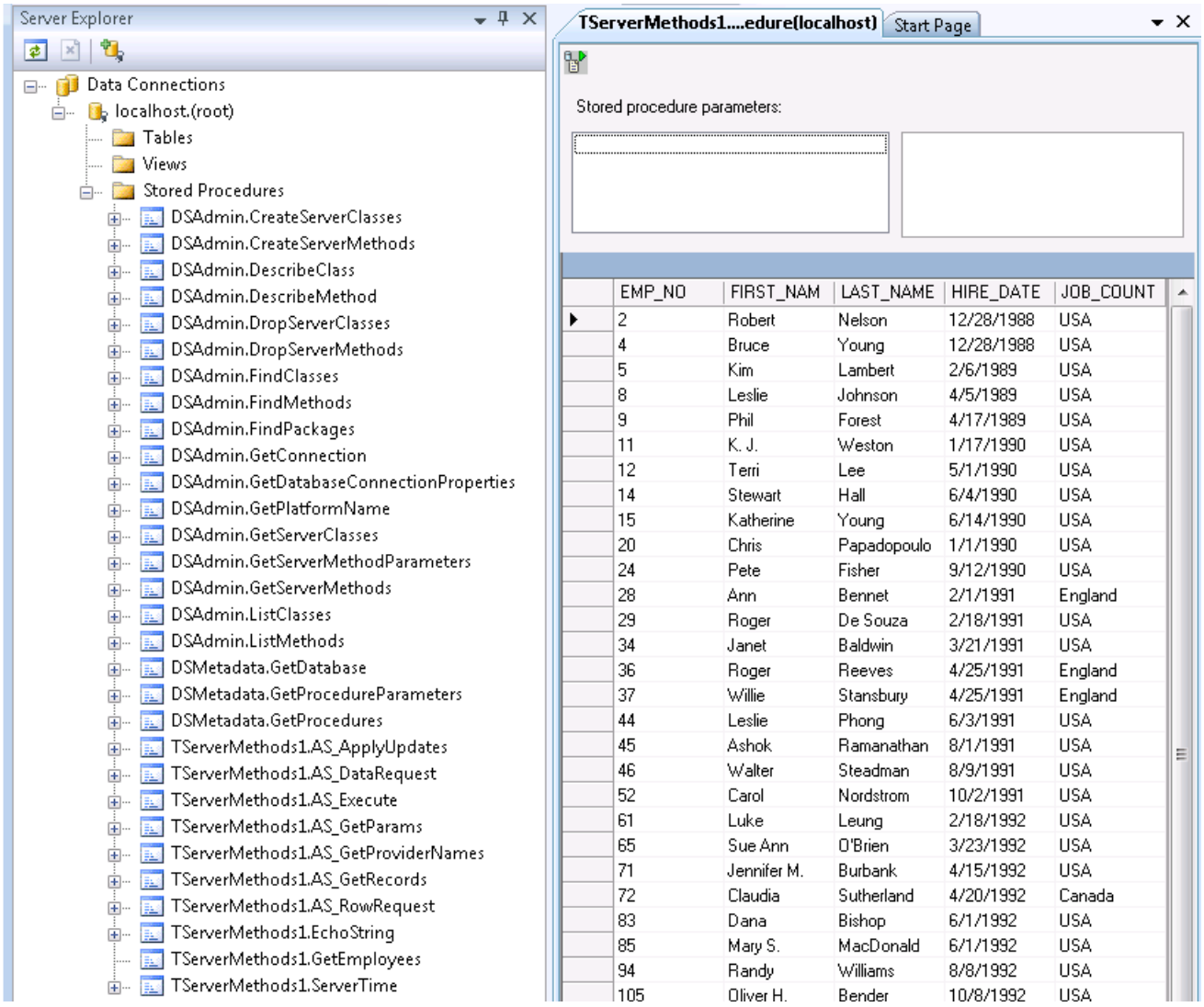


Nous pouvons maintenant tester certaines méthodes serveur depuis l'explorateur. Par exemple, en cliquant avec le bouton droit sur la méthode EchoString et en sélectionnant View Parameters. Cette opération affiche une nouvelle fenêtre où nous pouvons entrer une valeur pour le paramètre Value. Saisissons 42 ; cliquons avec le bouton droit dans la fenêtre et sélectionnons Execute. Cette opération exécute la méthode EchoString du serveur DataSnap et indique les résultats juste dessous la fenêtre des paramètres de procédure enregistrée :



Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

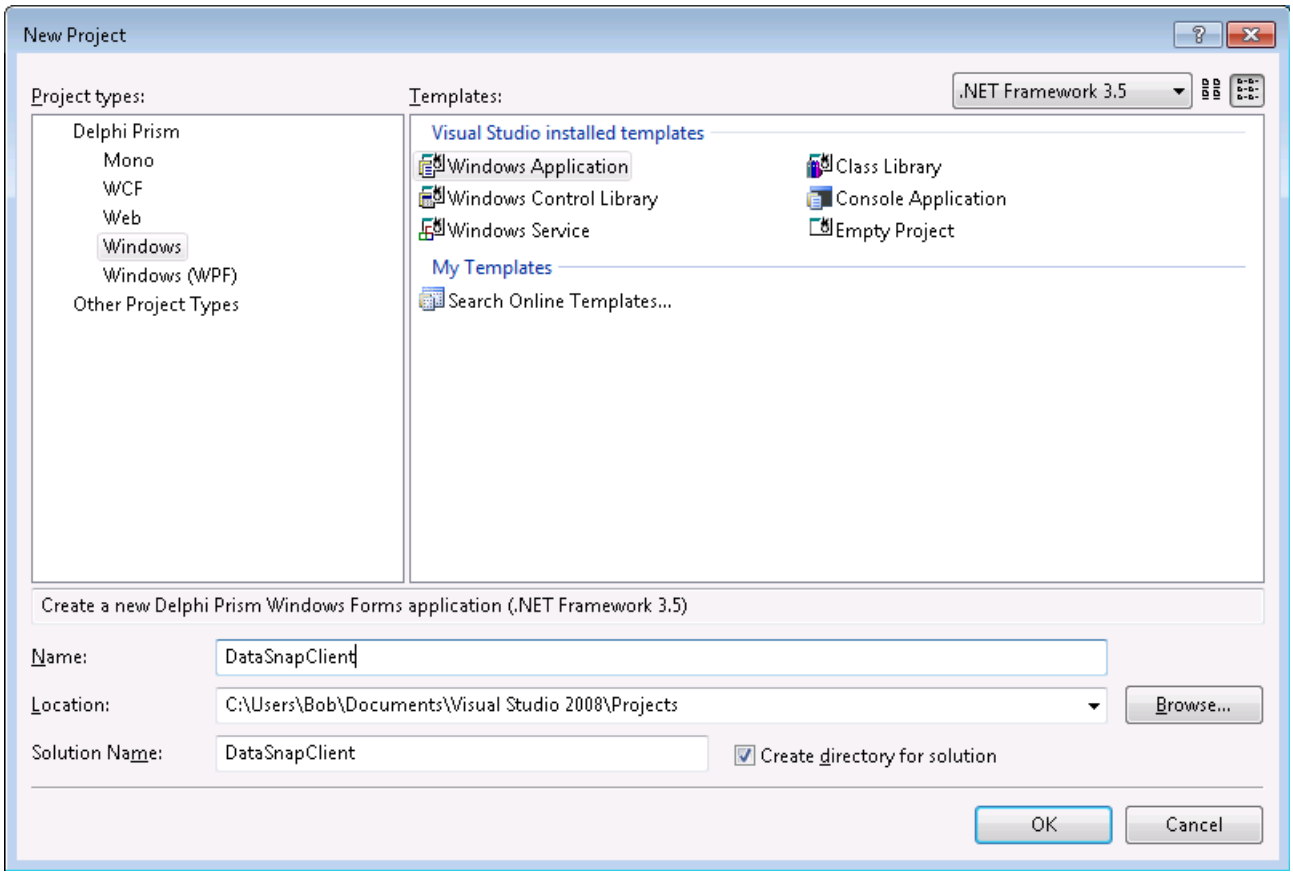
Il serait plus intéressant de savoir comment nous pouvons récupérer et utiliser les données de la table Employees en utilisant la méthode GetEmployees. Cette procédure stockée n'a pas de paramètres mais nous pouvons sélectionner la commande View Parameters qui retourne une liste vide de paramètres de procédure stockée. Cliquons à nouveau avec le bouton droit sur cette fenêtre et sélectionnons Execute. Cette fois, le résultat est le jeu complet d'enregistrements de la table Employee comme retourné par la méthode GetEmployees :



7.1. CLIENT WINFORMS

L'utilisation des méthodes serveur DataSnap dans l'explorateur est intéressante mais il est plus utile de les appeler depuis une application .NET. Pour ce dernier exemple choisissez Fichier | Nouveau projet pour démarrer l'assistant nouveau projet de Delphi Prism. Ceci donne un aperçu des cibles disponibles.
Dans le type de projet, sélectionnez Windows Application et modifiez le nom WindowsApplication1 en DataSnapClient.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.



Si vous cliquez sur OK, un nouveau projet DataSnapClient sera créé dans l'EDI Delphi Prism avec une unité Main.pas pour le formulaire principal.
Dans Server Explorer, sélectionnez la nouvelle connexion vers le serveur DataSnap que nous avons créée à la section précédente. L'explorateur de propriétés affiche les propriétés – et notamment ConnectionString qui doit se présenter comme suit :

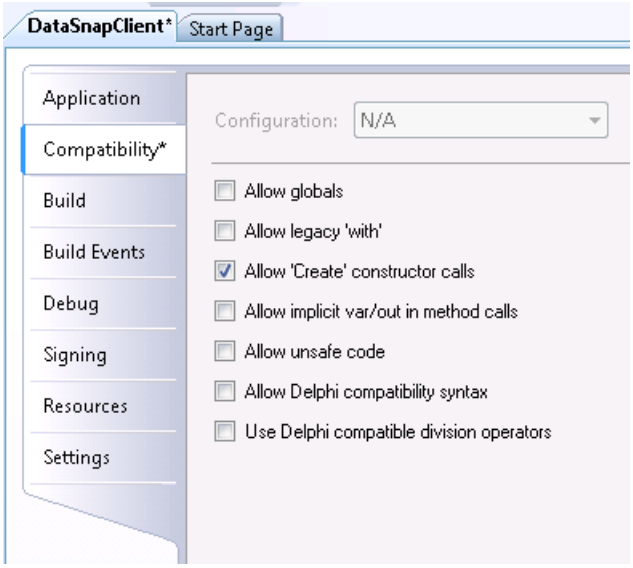
```
communicationprotocol=http;hostname=localhost;port=8080;dsauthenticationuser=Bob;ds
authenticationpassword=Swart
```

Cliquez avec le bouton droit sur le nœud de connexion aux données et sélectionnez Generate Client Proxy.
Cela génère un fichier ClientProxy1.pas avec la définition d'une classe dénommée TServerMethods1Client et un certain nombre de méthodes (EchoString, ServerTime et GetEmployees). Le snippet de définition de classe se présente comme suit :

```
TServerMethods1Client = class
public
constructor (ADBXConnection: TAdoDbxConnection);
constructor (ADBXConnection: TAdoDbxConnection; AInstanceOwner: Boolean);
function EchoString(Value: string): string;
function ServerTime: DateTime;
function GetEmployees: System.Data.IDataReader;
```

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

À part la classe proxy, plusieurs références ont été ajoutées au nœud References du projet : Borland.Data.AdoDbxClient et Borland.Data.DbxClientDriver.
Comme nous le voyons dans le snippet de TServerMethods1Client, cette classe a deux constructeurs : les deux avec un paramètre ADBXConnection et la deuxième avec un paramètre booléen AInstanceOwner. Cela signifie que nous devons appeler le constructeur avec un argument. Pour cela nous devons modifier les paramètres du projet. Cliquez avec le bouton droit sur le nœud DataSnapClient dans l'explorateur de solution et sélectionnez Propriétés. Dans la fenêtre, choisissez l'onglet Compatibilité et cochez l'option Allow Create constructor calls permettant d'appeler le constructeur Create passant les arguments au lieu d'utiliser le nouvel opérateur.



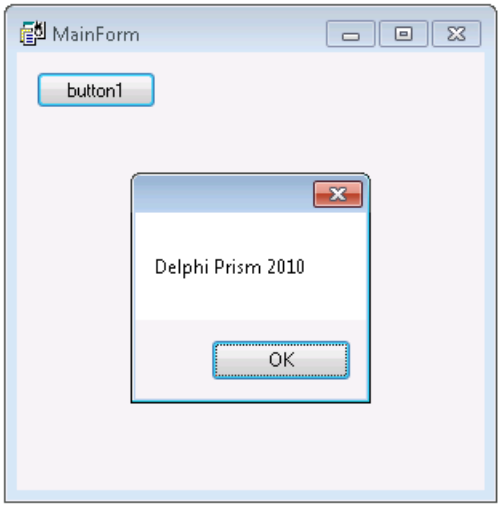
Nous pouvons maintenant retourner au formulaire principal et lui ajouter un bouton. Dans l'événement « Clic » du bouton, nous pouvons créer une connexion au serveur DataSnap et appeler une de ses méthodes.

```
method MainForm.button1_Click(sender: System.Object; e: System.EventArgs);
var
    Client: ClientProxy1.TServerMethods1Client;
    Connection: Borland.Data.TAdoDbxDatasnapConnection;
begin
    Connection := new Borland.Data.TAdoDbxDatasnapConnection();
    Connection.ConnectionString :=
    'communicationprotocol=http;hostname=localhost;port=8080;dsauthenticationuser=Bob;dsa
    enticationpassword=Swart';
    Connection.Open;
    try
          Client := ClientProxy1.TServerMethods1Client.Create(Connection);
          MessageBox.Show(
          Client.EchoString('Delphi Prism 2010'));
    finally
          Connection.Close;
    end;
```

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

```
end;
```

Le résultat est l'écho de Delphi Prism 2010 comme indiqué ci-dessous.



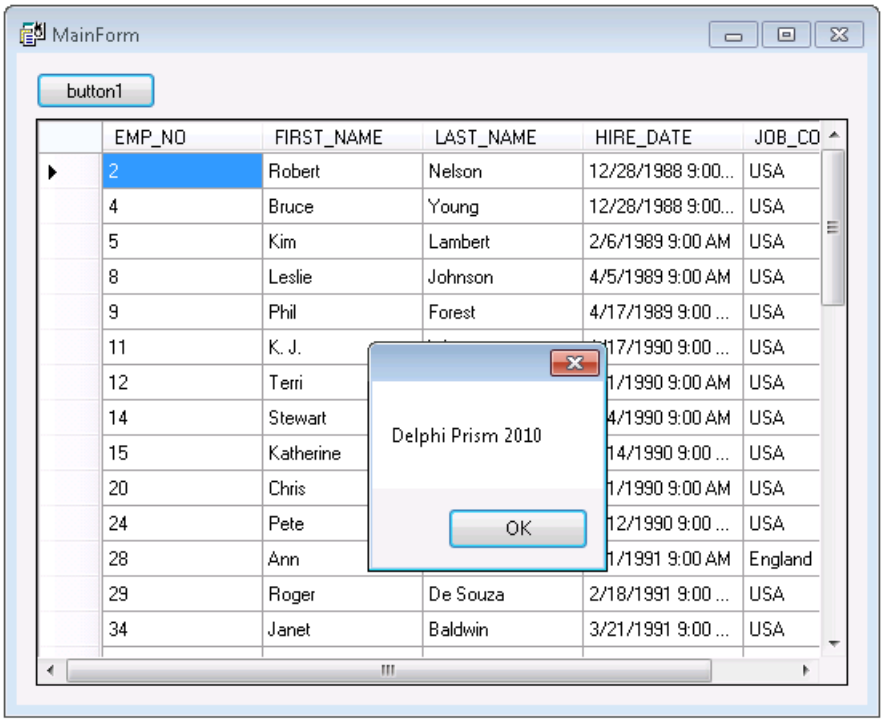
De la même façon, nous pouvons appeler la méthode GetEmployees et affecter les résultats à la vue DataGridView. Ceci pose un problème dans la mesure où GetEmployees retourne IDataReader (l'équivalent d'un résultat de TSQLDataSet) et pas un jeu DataSet ni une table DataTable. Nous devons donc écrire quelques lignes de code pour charger les résultats de GetEmployees dans une nouvelle table DataTable dans un jeu DataSet (l'équivalent de TClientDataSet du côté Win32).

```
method MainForm.button1_Click(sender: System.Object; e: System.EventArgs);
var
    Client: ClientProxy1.TServerMethods1Client;
    Connection: Borland.Data.TAdoDbxDatasnapConnection;
    Employees: System.Data.IDataReader;
    ds: System.Data.DataSet;
    dt: System.Data.DataTable;
begin
    Connection := new Borland.Data.TAdoDbxDatasnapConnection();
    Connection.ConnectionString :=
    'communicationprotocol=http;hostname=localhost;port=8080;dsauthenticationuser=Bob;dsa
    enticationpassword=Swart';
    Connection.Open;
    try
    Client := ClientProxy1.TServerMethods1Client.Create(Connection);
    Employees := Client.GetEmployees;
    ds := new DataSet();
    dt := new DataTable("DataSnap");
    ds.Tables.Add(dt);
    ds.Load(Employees, LoadOPtion.PreserveChanges, ds.Tables[0]);
    dataGridView1.DataSource := ds.Tables[0];
MessageBox.Show(
```

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

```
            Client.EchoString('Delphi Prism 2010'));
        finally
            Connection.Close;
        end;
    end;
```

Le résultat est les données suivantes présentées dans une vue en grille d'une application Delphi Prism WinForms démontrant que nous pouvons écrire des clients .NET légers pour se connecter aux serveurs DataSnap.



Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.

8. SYNTHÈSE

Nous avons démontré dans ce livre blanc que nous pouvons utiliser DataSnap où nous voulons (sous Windows avec une interface, un service ou une application console ou sur le Web avec une application CGI, ISAPI ou Débogueur d'application Web) ainsi que comme clients Win32 ou .NET et comme nous voulons en utilisant TCP/IP, HTTP avec authentification HTTP et sur option avec des filtres de compression, de cryptage, etc.

DataSnap 2010 offre des extensions et améliorations substantielles par rapport à DataSnap 2009 et des évolutions remarquables depuis les versions originales de DataSnap et MIDAS basées sur COM.

Bob Swart Bob Swart Training & Consultancy (eBob4)^

Embarcadero Technologies, Inc. est un éditeur leader de solutions primées pour les développeurs d'applications et les professionnels des bases de données pour les aider à concevoir et exécuter plus efficacement et rapidement leurs solutions – quels que soient les langages de programmation ou les plates-formes. 90 des 100 premières sociétés mondiales du classement Fortune et une communauté active de plus de 3 millions d'utilisateurs dans le monde font confiance aux outils d'Embarcadero pour maximiser leur productivité, réduire les coûts, simplifier la gestion du changement et de la conformité et accélérer l'innovation.

Les produits phares d'Embarcadero sont notamment : Embarcadero® Change Manager™, RAD Studio, DBArtisan®, Delphi®, ER/Studio®, JBuilder® et Rapid SQL®. La société Embarcadero a été créée en 1993 et son siège se situe à San Francisco ; l'entreprise est présente dans le monde entier et son site Web peut être consulté à l'adresse www.embarcadero.com.

Copyright © 2009 Bob Swart (aka Dr.Bob - www.drbob42.com). Tous droits réservés.